
Tamaas Documentation

Release 2.7.1+0.g1937901

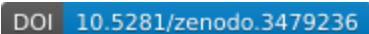
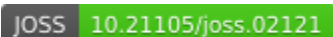
Lucas Frérot

Sep 20, 2023

TABLE OF CONTENTS:

1	Library overview	3
1.1	Tutorials	3
1.2	Citations	3
1.3	Changelog	4
1.4	Seeking help	4
1.5	Contribution	4
2	Quickstart	5
2.1	Installation from PyPI	5
2.2	Installation with Spack	5
2.3	Docker image	6
2.4	Singularity container	7
2.5	Installation from source	7
2.6	Building the docs	9
2.7	Running the contact pipe tools	9
2.8	Running the tests	9
3	Random rough surfaces	11
3.1	Generating a surface	11
3.2	Custom filter	12
3.3	Surface Statistics	13
4	Model and integral operators	15
4.1	Units in Tamaas	15
4.2	Model types	15
4.3	Model creation and basic functionality	16
4.4	Integral operators	17
4.5	Model dumpers	18
5	Solving contact	19
5.1	Elastic contact	19
5.2	Contact with adhesion	20
5.3	Tangential contact	21
5.4	Elasto-plastic contact	21
5.5	Custom Contact Solvers	22
6	Working with MPI	23
6.1	Transparent MPI context	23
6.2	MPI convenience methods	25
7	Examples	27

8	Performance	29
8.1	Parallelism	29
8.2	Integration algorithm	30
8.3	Computational methods & Citations	30
9	Frequently Asked Questions	33
9.1	Importing <code>tamaas</code> module gives a circular import error on Windows	33
9.2	What are the units in Tamaas?	33
9.3	Do contact solvers solve for a total force or an average pressure?	33
9.4	<code>scons</code> dev fails to install Tamaas with <code>externally-managed-environment</code> error	33
10	API Reference	35
10.1	Python API	35
10.2	C++ API	65
11	Indices and tables	189
	Python Module Index	191
	Index	193

DOI [10.5281/zenodo.3479236](https://doi.org/10.5281/zenodo.3479236)JOSS [10.21105/joss.02121](https://doi.org/10.21105/joss.02121)launch binder

Tamaas (from [سامت](#) meaning “contact” in Arabic and Farsi) is a high-performance rough-surface periodic contact code based on boundary and volume integral equations. The clever mathematical formulation of the underlying numerical methods (see e.g. [doi:10.1007/s00466-017-1392-5](https://doi.org/10.1007/s00466-017-1392-5) and [arxiv:1811.11558](https://arxiv.org/abs/1811.11558)) allows the use of the fast-Fourier Transform, a great help in achieving peak performance: Tamaas is consistently *two orders of magnitude faster* (and lighter) than traditional FEM! Tamaas is aimed at researchers and practitioners wishing to compute realistic contact solutions for the study of interface phenomena.

You can see Tamaas in action with our interactive [tutorials](#).

LIBRARY OVERVIEW

Tamaas is mainly composed of three components:

- Random surface generation procedures
- Model state objects and operators
- Contact solving procedures

These parts are meant to be independent as well as inter-operable. The first one provides an interface to several stochastic surface generation algorithms described in *Random rough surfaces*. The second provides an interface to a state object *Model* (and C++ class `tamaas::Model`) as well as integral operators based on the state object (see *Model and integral operators*). Finally, the third part provides contact solving routines that make use of the integral operators for performance (see *Solving contact*).

1.1 Tutorials

The following tutorial notebooks can also be used to learn about Tamaas:

- Elastic Contact ([live notebook](#), [notebook viewer](#))
- Rough Surfaces ([live notebook](#), [notebook viewer](#))

1.2 Citations

Tamaas is the fruit of a research effort. To give proper credit to its creators and the scientists behind the methods it implements, you can use the `tamaas.utils.publications()` function at the end of your python scripts. This function scans global variables and prints the relevant citations for the different capabilities of Tamaas used. Note that it may miss citations if some objects are not explicitly stored in named variables, so please examine the relevant publications in [doi:10.21105/joss.02121](https://doi.org/10.21105/joss.02121).

1.3 Changelog

The changelog can be consulted [here](#).

1.4 Seeking help

You can ask your questions on [gitlab](#) using this [form](#). If you do not have an account, you can create one [on this page](#).

1.5 Contribution

Contributions to Tamaas are welcome, whether they are code, bug, or documentation related.

1.5.1 Code

Please [fork](#) Tamaas and [submit your patch](#) as a merge request.

1.5.2 Bug reports

You can also contribute to Tamaas by reporting any bugs you find [here](#) if you have an account on [gitlab](#). Please read the [FAQ](#) before posting.

QUICKSTART

Here is a quick introduction to get you started with Tamaas.

2.1 Installation from PyPI

If you have a Linux system, or have installed the [Windows Subsystem for Linux](#), you can simply run `python3 -m pip install tamaas`. This installs the latest stable version of Tamaas from [PyPI](#). You can get the latest cutting-edge build of Tamaas with:

```
python3 -m pip install \
  --extra-index-url https://gitlab.com/api/v4/projects/19913787/packages/pypi/simple \
  tamaas
```

Note: To limit the number of statically linked dependencies in the wheel package, the builds that can be installed via PyPI or the GitLab package registry do not include parallelism or architecture specific optimizations. If you want to execute Tamaas in parallel, or want to improve performance, it is highly recommended that you install with Spack or compile from source with the instructions below.

2.2 Installation with Spack

If you have [Spack](#) installed (e.g. in an HPC environment), you can install Tamaas with the following:

```
spack install tamaas
```

This will install an MPI-enabled version of tamaas with Scipy solvers. The command `spack info tamaas` shows the build variants. To disable MPI, you should disable MPI in the FFTW variant:

```
spack install tamaas ^fftw~mpi
```

The Spack package for Tamaas allows installation from the repository with `spack install tamaas@master`.

You can also use Spack to create container recipes with `spack containerize`. Here's an example `spack.yaml` for Tamaas which creates an image with OpenMPI and NetCDF so that Tamaas can be executed in parallel within the container with [Singularity](#):

```
spack:
config:
  build_jobs: 2
```

(continues on next page)

(continued from previous page)

```
specs:
- matrix:
  - [py-mpi4py, py-netcdf4, tamaas@master+solvers]
  - ["^openmpi@4 fabrics=ofi,psm2,ucx schedulers=none"]
concretizer:
  unify: true
container:
  format: singularity
  images:
  os: ubuntu:20.04
  spack: develop
  os_packages:
  final:
    - gfortran
```

2.3 Docker image

We provide Docker images that are automatically built and pushed to the Gitlab registry for compatibility reasons (mainly with macOS). To get the latest image simply run:

```
docker pull registry.gitlab.com/tamaas/tamaas
# to rename for convenience
docker tag registry.gitlab.com/tamaas/tamaas tamaas
```

Then you can run scripts directly:

```
docker run -v $PWD:/app -t tamaas python3 /app/your_script.py
# or
docker run tamaas tamaas surface --sizes 16 16 --cutoffs 4 4 8 --hurst 0.8 | docker_
↪run -i tamaas tamaas contact 0.1 > results.npy
```

Warning: The provided Docker image and Dockerfile offer limited customization options and are hard-wired to use OpenMPI. If the use case/goal is to run in HPC environments, containers generated with Spack should be preferred, as they allow greater flexibility in the dependency selection and installed packages.

The Dockerfile at the root of the Tamaas repository can be used to build an image containing Tamaas with a full set of dependencies and customize the build options. Simply run:

```
docker build -t tamaas .
```

Tip: The following arguments can be passed to docker to customize the Tamaas build (with the `--build-arg` flag for `docker build`):

- `BACKEND`: parallelism model for loops
- `FFTW_THREADS`: parallelism model for FFTW3
- `USE_MPI`: compile an MPI-parallel version of Tamaas

See below for explanations.

2.4 Singularity container

A [singularity](#) container can be created from the docker image with:

```
singularity build tamaas.sif docker://registry.gitlab.com/tamaas/tamaas
```

To run the image with MPI:

```
mpirun singularity exec --home=$PWD tamaas.sif python3 <your_script>
```

2.5 Installation from source

First make sure the following dependencies are installed for Tamaas:

- a **C++ compiler** with full **C++14** and **OpenMP** support
- **SCons** (python build system)
- **FFTW3**
- **thrust** (1.9.2+)
- **boost** (pre-processor)
- **python 3+** with **numpy**
- **pybind11** (included as submodule)
- **expolit** (included as submodule)

Optional dependencies are:

- an MPI implementation
- **FFTW3** with MPI/threads/OpenMP (your pick) support
- **scipy** (for nonlinear solvers)
- **uvw**, **h5py**, **netCDF4** (for dumpers)
- **googletest** and **pytest** (for tests)
- **Doxygen** and **Sphinx** (for documentation)

Tip: On a HPC environment, use the following variables to specify where the dependencies are located:

- FFTW_ROOT
 - THRUST_ROOT
 - BOOST_ROOT
 - PYBIND11_ROOT
 - GTEST_ROOT
-

Note: You can use the provided Dockerfile to build an image with the external dependencies installed.

You should first clone the git repository with the submodules that are dependencies to tamaas ([pybind11](#) and [googletest](#)):

```
git clone --recursive https://gitlab.com/tamaas/tamaas.git
```

The build system uses [SCons](#). In order to compile Tamaas with the default options:

```
scons
```

After compiling a first time, you can edit the compilation options in the file `build-setup.conf`, or alternatively supply the options directly in the command line:

```
scons option=value [...]
```

To get a list of **all** build options and their possible values, you can run `scons -h`. You can run `scons -H` to see the SCons-specific options (among them `-j n` executes the build with `n` threads and `-c` cleans the build). Note that the build is aware of the `CXX` and `CXXFLAGS` environment variables.

Once compiled, you can install the python module with:

```
scons install prefix=/your/prefix
```

The above command automatically calls `pip` if it is installed. If want to install an editable package for development, the best practice is to run the following in a [virtual environment](#):

```
scons dev
```

This is equivalent to running `pip install -e build-release/python[all]`. You can check that everything is working fine with:

```
python3 -c 'import tamaas; print(tamaas)'
```

2.5.1 Important build options

Here are a selected few important compilation options:

build_type

Controls the type of build, which essentially changes the optimisation level (`-O0` for debug, `-O2` for profiling and `-O3` for release) and the amount of debug information. Default type is `release`. Accepted values are:

- `release`
- `debug`
- `profiling`

CXX

Compiler (uses the environment variable `CXX` by default).

CXXFLAGS

Compiler flags (uses the environment variable `CXXFLAGS` by default). Useful to add tuning flags (e.g. `-march=native -mtune=native` for GCC or `-xHOST` for Intel), or additional optimisation flags (e.g. `-flto` for link-time optimization).

backend

Controls the Thrust parallelism backend. Defaults to `omp` for OpenMP. Accepted values are:

- `omp`: OpenMP
- `cpp`: Sequential
- `tbb` (unsupported): Threading Building Blocks

fftw_threads

Controls the FFTW thread model. Defaults to `omp` for OpenMP. Accepted values are:

- `omp`: OpenMP
- `threads`: PThreads
- `none`: Sequential

use_mpi

Activates MPI-parallelism (boolean value).

More details on the above options can be found in [Performance](#).

2.6 Building the docs

Documentation is built using `scons doc`. Make sure to have the correct dependencies installed (they are already provided in the Docker image).

2.7 Running the contact pipe tools

Once installed, the python package provides a *tamaas* command, which defines three subcommands that can be used to explore the mechanics of elastic rough contact:

- `surface` generates randomly rough surfaces (see [Random rough surfaces](#))
- `contact` solves an elastic contact problem with a surface read from `stdin` (see [Solving contact](#))
- `plot` plots the surface tractions and displacements read from `stdin`

Here's a sample command line for you to try out:

```
tamaas surface --cutoffs 2 4 64 --size 512 512 --hurst 0.8 | tamaas contact 1 |  
↪tamaas plot
```

Check out the help of each command (e.g. `tamaas surface -h`) for a description of the arguments.

2.8 Running the tests

You need to activate the `build_tests` option to compile the tests:

```
scons build_tests=True
```

Tests can then be run with the `scons test` command. Make sure you have `pytest` installed.

RANDOM ROUGH SURFACES

The generation of stochastically rough surfaces is controlled in Tamaas by two abstract classes: `tamaas::SurfaceGenerator` and `tamaas::Filter`. The former provides access lets the user set the surface sizes and random seed, while the latter encodes the information of the spectrum of the surface. Two surface generation methods are provided:

- `tamaas::SurfaceGeneratorFilter` implements a Fourier filtering algorithm (see [Hu & Tonder](#)),
- `tamaas::SurfaceGeneratorRandomPhase` implements a random phase filter.

Both of these rely on a `tamaas::Filter` object to provide the filtering information (usually power spectrum density coefficients). Tamaas provides two such classes and allows for Python subclassing:

- `tamaas::Isopowerlaw` provides a roll-off powerlaw,
- `tamaas::RegularizedPowerlaw` provides a powerlaw with a regularized rolloff.

Tamaas also provides routines for surface statistics.

3.1 Generating a surface

Let us now see how to generate a surface. First create a filter object and set the surface sizes:

```
import tamaas as tm

# Create spectrum object
spectrum = tm.Isopowerlaw2D()

# Set spectrum parameters
spectrum.q0 = 4
spectrum.q1 = 4
spectrum.q2 = 32
spectrum.hurst = 0.8
```

The spectrum object can be queried for information, such as the root-mean-square of heights, the various statistical moments, the spectrum bandwidth, etc. Then we create a generator object and build the random surface:

```
generator = tm.SurfaceGeneratorFilter2D([128, 128])
generator.spectrum = spectrum
generator.random_seed = 0

surface = generator.buildSurface()
```

Important: The surface object is a `numpy.ndarray` wrapped around internal memory in the generator object, so a subsequent call to `buildSurface` may change its content. Note that if generator goes out of scope its memory

will not be freed if there is still a live reference to the surface data.

Important: If ran in an MPI context, the constructor of `SurfaceGeneratorFilter2D` (and others) expects the *global* shape of the surface. The shape can also be changed with `generator.shape = [64, 64]`.

It is common to normalize a surface after it has been generated so that one can scale the surface to a desired statistic, e.g. to specify the root-mean-square of slopes:

```
rms_slopes = 0.25
surface /= iso.rmsSlopes()
surface *= rms_slopes

# Compute root-mean-square of slopes in Fourier domain
print(tm.Statistics2D.computeSpectralRMSSlope(surface))
# Compute root-mean-square of slopes with finite differences
# (introduces discretization error)
print(tm.Statistics2D.computeFDRMSSlope(surface))
```

Note: The spectrum object gives the expected value of surface statistics, but the corresponding value for a surface realization can differ (particularly if the `SurfaceGeneratorFilter2D` is used). One could normalize a surface with the *actual* statistic, but this leads to **biased** quantities when a representative sample of surfaces is used.

3.2 Custom filter

Tamaas provides several classes that can be derived directly with Python classes, and `tamaas::Filter` is one of them. Since it provides a single pure virtual method `computeFilter`, it is easy to write a sub-class. Here we implement a class that takes a user-defined auto-correlation function and implements the `computeFilter` virtual function:

```
import numpy

class AutocorrelationFilter(tm.Filter2D):
    def __init__(self, autocorrelation):
        tm.Filter2D.__init__(self)
        self.autocorrelation = autocorrelation.copy()

    def computeFilter(self, filter_coefficients):
        shifted_ac = numpy.fft.ifftshift(self.autocorrelation)

        # Fill in the PSD coefficients
        filter_coefficients[...] = numpy.sqrt(np.fft.rfft2(shifted_ac))
        # Normalize
        filter_coefficients[...] *= 1 / numpy.sqrt(self.autocorrelation.size)
```

Here `filter_coefficients` is also a `numpy.ndarray` and is therefore easily manipulated. The creation of the surface then follows the same pattern as previously:

```
# Create spectrum object
autocorrelation = ... # set your desired autocorrelation
spectrum = AutocorrelationFilter(autocorrelation)
```

(continues on next page)

(continued from previous page)

```

generator = tm.SurfaceGenerator2D()
generator.shape = autocorrelation.shape
generator.spectrum = spectrum

surface = generator.buildSurface()

```

The lifetime of the spectrum object is associated to the generator when `setSpectrum` is called.

3.3 Surface Statistics

Tamaas provides the C++ class `tamaas::Statistics` and its wrapper `Statistics2D` to compute statistics on surfaces, including:

- power spectrum density
- autocorrelation
- spectrum moments
- root-mean-square of heights $\sqrt{\langle h^2 \rangle}$
- root-mean-square of slopes (computed in Fourier domain) $\sqrt{\langle |\nabla h|^2 \rangle}$

All these quantities are computed in a discretization-independent manner: increasing the number of points in the surface should not drastically change the computed values (for a given spectrum). This allows to refine the discretization as much as possible to approximate a continuum. Note that the autocorrelation and PSD are fft-shifted. Here is a sample code plotting the PSD and autocorrelation:

```

psd = tm.Statistics2D.computePowerSpectrum(surface)
psd = numpy.fft.fftshift(psd, axes=0) # Shifting only once axis because of R2C_
↳ transform

import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

plt.imshow(psd.real, norm=LogNorm())

acf = tm.Statistics2D.computeAutocorrelation(surface)
acf = numpy.fft.fftshift(acf)

plt.figure()
plt.imshow(acf)

plt.show()

```

See `examples/statistics.py` for more usage examples of statistics.

MODEL AND INTEGRAL OPERATORS

The class `tamaas::Model` (and its counterpart `Model`) is both a central class in Tamaas and one of the simplest. It mostly serves as holder for material properties, fields and integral operators, and apart from a linear elastic behavior does not perform any computation on its own.

4.1 Units in Tamaas

All quantities in Tamaas are unitless. To put real units onto them, one can use the following equation, which must have a consistent set of units:

$$\frac{u}{L} = \frac{\pi}{k} \frac{p}{\frac{E}{1-\nu^2}}.$$

It relates the surface displacement u to the pressure p , with the Young's modulus E , the Poisson coefficient ν and the *horizontal* physical size of the system L (i.e. the horizontal period of the system). The wavenumber k is adimensional. This means that L is the natural unit for lengths (which can be specified when creating the `Model` object), and $E^* := E/(1 - \nu^2)$ is the natural unit for pressures (which can be accessed with `model.E_star`). All other units (areas, forces, energies, etc.) are derived from these two units.

In general, it is a good idea to work with a unit set that gives numerical values of the same order of magnitude to minimize floating point errors in algebraic operations. While this is not always possible, because in elastic contact we have $u \sim h_{\text{rms}}$ and $p \sim h'_{\text{rms}} E^*$, which are dependent, one should avoid using meters if expected lengths are nanometers for example.

4.2 Model types

`tamaas::Model` has a concrete subclass `tamaas::ModelTemplate` which implements the model function for a given `tamaas::model_type`:

`tamaas::basic_2d`

Model type used in normal frictionless contact: traction and displacement are 2D fields with only one component.

`tamaas::surface_2d`

Model type used in frictional contact: traction and displacement are 2D fields with three components.

`tamaas::volume_2d`

Model type used in elastoplastic contact: tractions are the same as with `tamaas::surface_2d` but the displacement is a 3D field.

The enumeration values suffixed with `_1d` are the one dimensional (line contact) counterparts of the above model types. The domain physical dimension and number of components are encoded in the class `tamaas::model_type_traits`.

4.3 Model creation and basic functionality

The instantiation of a *Model* requires the model type, the physical size of the system, and the number of points in each direction:

```
physical_size = [1., 1.]
discretization = [512, 512]
model = tm.Model(tm.model_type.basic_2d, physical_size, discretization)
```

Warning: For models of type `volume_*d`, the first component of the `physical_size` and `discretization` arrays corresponds to the depth dimension (z in most cases). For example:

```
tm.Model(tm.model_type.basic_2d, [0.3, 1, 1], [64, 81, 81])
```

creates a model of depth 0.3 and surface size 1^2 , while the number of points is 64 in depth and 81^2 on the surface. This is done for data contiguity reasons, as we do discrete Fourier transforms in the horizontal plane.

Note: If ran in an MPI context, the constructor to *Model* expects the *global* system sizes and discretization of the model.

Tip: Deep copies of model objects can be done with the `copy.deepcopy()` function:

```
import copy

model_copy = copy.deepcopy(model)
```

Note that it only copies fields, not operators or dumpers.

4.3.1 Model properties

The properties `E` and `nu` can be used to set the Young's modulus and Poisson ratio respectively:

```
model.E = 1
model.nu = 0.3
```

4.3.2 Fields

Fields can be easlily accessed with the `[]` operator, similar to Python's dictionaries:

```
surface_traction = model['traction']
# surface_traction is a numpy array
```

To know what fields are available, you can call the `list` function on a model (`list(model)`). You can add new fields to a model object with the `[]` operator: `model['new_field'] = new_field`, which is convenient for dumping fields that are computed outside of Tamaas.

Note: The fields `traction` and `displacement` are always registered in models, and are accessible via `model.traction` and `model.displacement`.

A model can also be used to compute stresses from a strain field:

```
import numpy

strain = numpy.zeros(model.shape + [6]) # Mandel--Voigt notation
stress = numpy.zeros_like(strain)

model.applyElasticity(stress, strain)
```

Tip: `print(model)` gives a lot of information about the model: the model type, shape, registered fields, and more!

4.4 Integral operators

Integral operators are a central part of Tamaas: they are carefully designed for performance in periodic system. When a `Model` object is used with contact solvers or with a residual object (for plasticity), the objects using the model register integral operators with the model, so the user typically does not have to worry about creating integral operators.

Integral operators are accessed through the `operators` property of a model object. The `[]` operator gives access to the operators, and `list(model.operators)` gives the list of registered operators:

```
# Accessing operator
elasticity = model.operators['hooke']

# Applying operator
elasticity(strain, stress)

# Print all registered operators
print(list(model.operators))
```

Note: At model creation, these operators are automatically registered:

- `hooke`: Hooke's elasticity law
- `von_mises`: computes Von Mises stresses
- `deviatoric`: computes the deviatoric part of a stress tensor
- `eigenvalues`: computes the eigenvalues of a symmetric tensor field

Westergaard operators are automatically registered when `solveNeumann` or `solveDirichlet` are called.

4.5 Model dumpers

The submodule *tamaas.dumpers* contains a number of classes to save model data into different formats:

UVWDumper

Dumps a model to **VTK** format. Requires the **UVW** python package which you can install with pip:

```
pip install uvw
```

This dumper is made for visualization with VTK based software like **Paraview**.

NumpyDumper

Dumps a model to a compressed Numpy file.

H5Dumper

Dumps a model to a compressed **HDF5** file. Requires the **h5py** package. Saves separate files for each dump of a model.

NetCDFDumper

Dumps a model to a compressed **NetCDF** file. Requires the **netCDF4** package. Saves sequential dumps of a model into a single file, with the `frame` dimension containing the model dumps.

The dumpers are initialized with a basename and the fields that you wish to write to file (optionally you can set `all_fields` to `True` to dump all fields in the model). By default, each write operation creates a new file in a separate directory (e.g. *UVWDumper* creates a `paraview` directory). To write to a specific file you can use the *dump_to_file* method. Here is a usage example:

```
from tamaas.dumpers import UVWDumper, H5Dumper

# Create dumper
uvw_dumper = UVWDumper('rough_contact_example', 'stress', 'plastic_strain')

# Dump model
uvw_dumper << model

# Or alternatively
model.addDumper(H5Dumper('rough_contact_archive', all_fields=True))
model.addDumper(uvw_dumper)
model.dump()
```

The last `model.dump()` call will trigger all dumpers. The resulting files will have the following hierarchy:

```
./paraview/rough_contact_example_0000.vtr
./paraview/rough_contact_example_0001.vtr
./hdf5/rough_contact_archive_0000.h5
```

Important: Currently, only *H5Dumper* and *NetCDFDumper* support parallel output with MPI.

SOLVING CONTACT

The resolution of contact problems is done with classes that inherit from `tamaas:ContactSolver`. These usually take as argument a `tamaas:Model` object, a surface described by a `tamaas:Grid` or a 2D numpy array, and a tolerance. We will see the specificities of the different solver objects below.

5.1 Elastic contact

The most common case is normal elastic contact, and is most efficiently solved with `tamaas:PolonskyKeerRey`. The advantage of this class is that it combines two algorithms into one. By default, it considers that the contact pressure field is the unknown, and tries to minimize the complementary energy of the system under the constraint that the mean pressure should be equal to the value supplied by the user, for example:

```
# ...
solver = tm.PolonskyKeerRey(model, surface, 1e-12)
solver.solve(1e-2)
```

Here the average pressure is $1e-2$. The solver can also be instantiated by specifying the the constraint should be on the mean gap instead of mean pressure:

```
solver = tm.PolonskyKeerRey(model, surface, 1e-12, constraint_type=tm.PolonskyKeerRey.
    ↪ gap)
solver.solve(1e-2)
```

The contact problem is now solved for a mean gap of $1e-2$. Note that the choice of constraint affects the performance of the algorithm.

5.1.1 Computing solutions for loading sequence

The module `tamaas.utils` defines a convenience function `load_path`, which generates solution models for a sequence of loads. This allows lazy evaluation and reduces boiler-plate:

```
from tamaas.utils import load_path

loads = np.linspace(0.01, 0.1, 10)
for model in load_path(solver, loads):
    ... # do some computation on model, e.g. compute contact clusters
```

The function `load_path` accepts the following optional arguments:

verbose

Prints solver output (i.e. iteration, cost function and error)

callback

A function to call after each solve, before the next load step. Useful for dumping model in generator expressions.

5.2 Contact with adhesion

The second algorithm hidden in `tamaas::PolonskyKeerRey` considers the **gap** to be the unknown. The constraint on the mean value can be chosen as above:

```
solver = tm.PolonskyKeerRey(model, surface, 1e-12,
                             primal_type=tm.PolonskyKeerRey.gap,
                             constraint_type=tm.PolonskyKeerRey.gap)
solver.solve(1e-2)
```

The advantage of this formulation is to be able to solve adhesion problems (Rey et al.). This is done by adding a term to the potential energy functional that the solver tries to minimize:

```
adhesion_params = {
    "rho": 2e-3,
    "surface_energy": 2e-5
}

adhesion = tm.ExponentialAdhesionFunctional(surface)
adhesion.setParameters(adhesion)
solver.addFunctionalterm(adhesion)

solver.solve(1e-2)
```

Custom classes can be used in place of the example term here. One has to inherit from `tamaas::Functional`:

```
import numpy

class AdhesionPython(tm.Functional):
    """
    Functional class that extends a C++ class and implements the virtual
    methods
    """

    def __init__(self, rho, gamma):
        super().__init__(self)
        self.rho = rho
        self.gamma = gamma

    def computeF(self, gap, pressure):
        return -self.gamma * numpy.sum(np.exp(-gap / self.rho))

    def computeGradF(self, gap, gradient):
        gradient += self.gamma * numpy.exp(-gap / self.rho) / self.rho
```

This example is actually equivalent to `tamaas::functional::ExponentialAdhesionFunctional`.

5.3 Tangential contact

For frictional contact, several solver classes are available. Among them, `tamaas::Condat` is able to solve a coupled normal/tangential contact problem regardless of the material properties. It however solves an associated version of the Coulomb friction law. In general, the Coulomb friction used in contact makes the problem ill-posed.

Solving a tangential contact problem is not much different from normal contact:

```
mu = 0.3 # Friction coefficient
solver = tm.Condat(model, surface, 1e-12, mu)
solver.max_iter = 5000 # The default of 1000 may be too little
solver.solve([1e-2, 0, 1e-2]) # 3D components of applied mean pressure
```

5.4 Elasto-plastic contact

For elastic-plastic contact, one needs three different solvers: an elastic contact solver like the ones described above, a non-linear solver and a coupling solver. The non-linear solvers available in Tamaas are implemented in python and inherit from the C++ class `tamaas::EPSolver`. They make use of the non-linear solvers available in scipy:

DFSANESolver

Implements an interface to `scipy.optimize.root()` with the DFSANE method.

NewtonKrylovSolver

Implements an interface to `scipy.optimize.newton_krylov()`.

These solvers require a residual vector to cancel, the creation of which is handled with `tamaas::ModelFactory`. Then, an instance of `tamaas::EPICSolver` is needed to couple the contact and non-linear solvers for the elastic-plastic contact problem:

```
import tamaas as tm

from tamaas.nonlinear_solvers import DFSANESolver

# Definition of modeled domain
model_type = tm.model_type.volume_2d
discretization = [32, 51, 51] # Order: [z, x, y]
flat_domain = [1, 1]
system_size = [0.5] + flat_domain

# Setup for plasticity
material = tm.materials.IsotropicHardening(model,
                                           sigma_y=0.1 * model.E,
                                           hardening=0.01 * model.E)

residual = tm.Residual(model, material)
epsolver = DFSANESolver(residual)

# ...

csolver = tm.PolonskyKeerRey(model, surface, 1e-12)

epic = tm.EPICSolver(csolver, epsolver, 1e-7, relaxation=0.3)
epic.solve(1e-3)

# or use an accelerated scheme (which can sometimes not converge)

epic.acceleratedSolve(1e-3)
```

By default, `tamaas::EPICSolver::solve()` uses a relaxed fixed point. It can be tricky to make it converge: you need to decrease the relaxation parameter passed as argument of the constructor, but this also hinders the convergence rate. The function `tamaas::EPICSolver::acceleratedSolve()` does not require the tweaking of a relaxation parameter, so it can be faster if the latter does not have an optimal value. However, it is not guaranteed to converge.

Finally, during the first iterations, the fixed point error will be large compared to the error of the non-linear solver. It can improve performance to have the tolerance of the non-linear solver (which is the most expensive step of the fixed point solve) decrease over the iterations. This can be achieved with the decorator class `ToleranceManager`:

```
from tamaas.nonlinear_solvers import ToleranceManager, DFSANESolver

@ToleranceManager(start=1e-2,
                  end=1e-9,
                  rate=1 / 3)
class Solver(DFSANESolver):
    pass

# ...

epsolver = Solver(residual)

# or

epsolver = ToleranceManager(1e-2, 1e-9, 1/3)(DFSANESolver)(residual)
```

5.5 Custom Contact Solvers

The `tamaas::ContactSolver` class can be derived in Python so that users can interface with Scipy's `scipy.optimize.minimize()` function or PETSc's solvers accessible in the Python [interface](#). See [examples/scipy_penalty.py](#) for an example on how to proceed.

WORKING WITH MPI

Distributed memory parallelism in Tamaas is implemented with [MPI](#). Due to the bottleneck role of the fast-Fourier transform in Tamaas' core routines, the data layout of Tamaas is that of [FFTW](#). Tamaas is somewhat affected by limitations of FFTW, and MPI only works on systems with a 2D boundary, i.e. `basic_2d`, `surface_2d` and `volume_2d` model types (which are the most important anyways, since rough contact mechanics can yield different scaling laws in 1D).

MPI support in Tamaas is still experimental, but the following parts are tested:

- Rough surface generation
- Surface statistics computation
- Elastic normal contact
- Elastic-plastic contact (with [DFSANECXXSolver](#))
- Dumping models with [H5Dumper](#) and [NetCDFDumper](#).

Tip: One can look at `examples/plasticity.py` for a full example of an elastic-plastic contact simulation that can run in MPI.

6.1 Transparent MPI context

Some parts of Tamaas work transparently with MPI and no additional work or logic is needed.

Warning: `MPI_Init()` is automatically called when importing the `tamaas` module in Python. While this works transparently most of the time, in some situations, e.g. in Singularity containers, the program can hang if `tamaas` is imported first. It is therefore advised to run `from mpi4py import MPI` *before* `import tamaas` to avoid issues.

6.1.1 Creating a model

The following snippet creates a model whose global shape is [16, 2048, 2048]:

```
import tamaas as tm

model = tm.Model(tm.model_type.volume_2d,
                 [0.1, 1, 1], [16, 2048, 2048])
print(model.shape, model.global_shape)
```

Running this code with `mpirun -np 3` will print the following (not necessarily in this order):

```
[16, 683, 2048] [16, 2048, 2048]
[16, 682, 2048] [16, 2048, 2048]
[16, 683, 2048] [16, 2048, 2048]
```

Note that the partitioning occurs on the x dimension of the model (see below for more information on the data layout imposed by FFTW).

6.1.2 Creating a rough surface

Similarly, rough surface generators expect a global shape and return the partitioned data:

```
iso = tm.Isopowerlaw2D()
iso.q0, iso.q1, iso.q2, iso.hurst = 4, 4, 32, .5
gen = tm.SurfaceGeneratorRandomPhase2D([2048, 2048])
gen.spectrum = iso

surface = gen.buildSurface()
print(surface.shape, tm.mpi.global_shape(surface.shape))
```

With `mpirun -np 3` this should print:

```
(682, 2048) [2048, 2048]
(683, 2048) [2048, 2048]
(683, 2048) [2048, 2048]
```

Handling partitioning edge cases

Under certain conditions, FFTW may assign to one or more processes a size of zero to the x dimension of the model. If that happens, the surface generator will raise a runtime error, which causes a deadlock because it does not exit the processes with zero data. The correct way to handle this edge case is:

```
from mpi4py import MPI

try:
    gen = tm.SurfaceGeneratorRandomPhase2D([128, 128])
except RuntimeError as e:
    print(e)
    MPI.COMM_WORLD.Abort(1)
```

This will correctly kill all processes. Alternatively, `os._exit()` can be used, but one should avoid `sys.exit()`, as it kills the process by raising an exception, which still results in a deadlock.

6.1.3 Computing statistics

With a model's data distributed among independent process, computing global properties, like the true contact area, must be done in a collective fashion. This is transparently handled by the *Statistics* class, e.g. with:

```
contact = tm.Statistics2D.contact(model.traction)
```

This gives the correct contact fraction, whereas something like `np.mean(model.traction > 0)` will give a different result on each processor.

6.1.4 Nonlinear solvers

The only nonlinear solver (for plastic contact) that works with MPI is *DFSANECXXSolver*, which is a C++ implementation of *DFSANESolver* that works in an MPI context.

Note: Scipy and Numpy use optimized BLAS routines for array operations, while Tamaas does not, which results in *serial* performance of the C++ implementation of the DF-SANE algorithm being lower than the Scipy version.

6.1.5 Dumping models

The only dumpers that properly works in MPI are the *H5Dumper* and *NetCDFDumper*. Output is then as simple as:

```
from tamaas.dumpers import H5Dumper

H5Dumper('output', all_fields=True) << model
```

This is useful for doing post-processing separately from the main simulation: the post-processing can then be done in serial.

6.2 MPI convenience methods

Not every use case can be handled transparently, but although adapting existing scripts to work in an MPI context can require some work, especially if said scripts rely on numpy and scipy for pre- and post-processing (e.g. constructing a parabolic surface for hertzian contact, computing the total contact area), the module *mpi* provides some convenience functions to make that task easier. The functions *mpi.scatter* and *mpi.gather* can be used to scatter/gather 2D data using the partitioning scheme expected from FFTW (see figure below). The functions *mpi.rank* and *mpi.size* are used to determine the local process rank and the total number of processes respectively.

If finer control is needed, the function *mpi.local_shape* gives the 2D shape of the local data if given the global 2D shape (its counterpart *mpi.global_shape* does the exact opposite), while *mpi.local_offset* gives the offset of the local data in the global *x* dimension. These two functions mirror FFTW's own data distribution functions.

The *mpi* module also contains a function *sequential* whose return value is meant to be used as a context manager. Within the sequential context the default communicator is `MPI_COMM_SELF` instead of `MPI_COMM_WORLD`.

For other MPI functionality not covered by Tamaas that may be required, one can use *mpi4py*, which in conjunction with the methods in *mpi* should handle just about any use case.

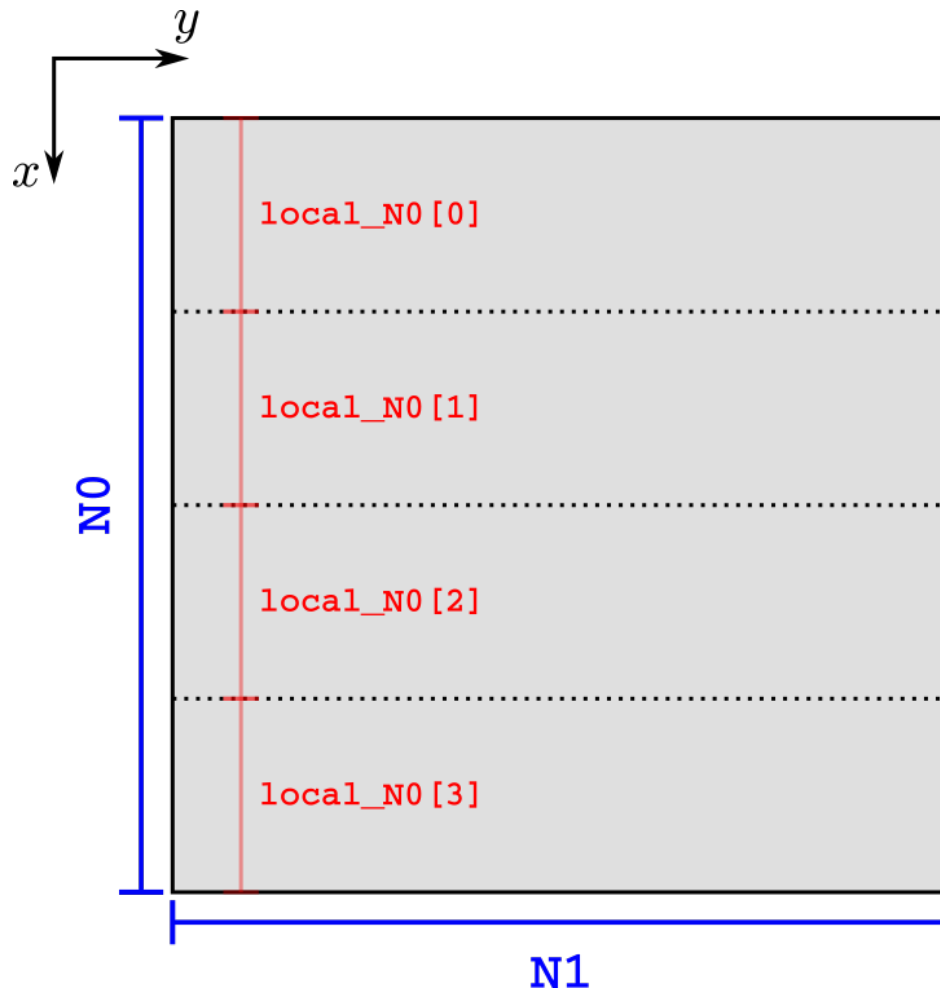


Fig. 1: 2D Data distribution scheme from FFTW. $N0$ and $N1$ are the number of points in the x and y directions respectively. The array `local_N0`, indexed by the process rank, give the local size of the x dimension. The `local_offset` function gives the offset in x for each process rank.

EXAMPLES

The directory `examples/` in Tamaas' root repository contains example scripts dedicated to various aspects of Tamaas:

`statistics.py`

This script generates a rough surface and computes its power spectrum density as well as its autocorrelation function.

`rough_contact.py`

This script generates a rough surface and solves an adhesion-less elastic contact problem.

`adhesion.py`

This script solves a rough contact problem with an exponential energy functional for adhesion. It also shows how to derive an energy functional in python.

`saturation.py`

This script solves a saturated contact problem (i.e. pseudo-plasticity) with a rough surface.

`stresses.py`

This script solves an equilibrium problem with an eigenstrain distribution and a surface traction distribution and writes the output to a VTK file. It demonstrates how the integral operators that Tamaas uses internally for elastic-plastic contact can be used directly in Python.

`plasticity.py`

This script solves an elastoplastic Hertz contact problem with three load steps and writes the result to VTK files.

`scipy_penalty.py`

This script shows how to implement a contact solver in python that uses the contact functionals of Tamaas. Here we use Scipy's `scipy.optimize.minimize()` function to solve a contact problem with penalty.

PERFORMANCE

8.1 Parallelism

Tamaas implements shared-memory parallelism using `thrust`. The Thrust backend can be controlled with the following values of the `backend` build option:

omp

Thrust uses its OpenMP backend (the default). The number of threads is controlled by OpenMP.

cpp

Thrust does not run in threads (i.e. sequential). This is the recommended option if running multiple MPI tasks.

tbb

Thrust uses its `TBB` backend. Note that this option is not fully supported by Tamaas.

Tip: When using the OpenMP or TBB backend, the number of threads can be manually controlled by the `initialize` function. When OpenMP is selected for the backend, the environment variable `OMP_NUM_THREADS` can also be used to set the number of threads.

FFTW has its own system for thread-level parallelism, which can be controlled via the `fftw_threads` option:

none

FFTW does not use threads.

threads

FFTW uses POSIX/Win32 threads for parallelism.

omp

FFTW uses OpenMP.

Note: As with the Thrust backend, the number of threads for FFTW can be controlled with `initialize`.

Finally, the boolean variable `use_mpi` controls whether Tamaas is compiled with MPI-parallelism. If yes, Tamaas will be linked against `libfftw3_mpi` regardless of the thread model.

Important: Users wary of performance should use MPI, as it yields remarkably better scaling properties than the shared memory parallelism models. Care should also be taken when compiling with both OpenMP and MPI support: setting the number of threads to more than one in an MPI context can decrease performance.

8.2 Integration algorithm

In its implementation of the volume integral operators necessary for elastic-plastic solutions, Tamaas differentiates two way of computing the intermediate integral along z in the partial Fourier domain:

- Cutoff integration: because the intermediate integral involves kernels of the form $\exp(q(x - y))$, it is easy to truncate the integral when x and y are far apart, especially for large values of q . This changes the complexity of the intermediate integral from $O(N_1 N_2 N_3^2)$ (the naive implementation) to $O(\sqrt{N_1^2 + N_2^2} N_3^2)$.
- Linear integration: this method relies on a separation of variables $\exp(q(x - y)) = \exp(qx) \cdot \exp(-qy)$. This allows to break the dependency in N_3^2 of the number of operations, so that the overall complexity of the intermediate integral is $O(N_1 N_2 N_3)$.

Details on both algorithms can be found in¹. Tamaas uses linear integration by default because it is faster in many cases without introducing a truncation error. Unfortunately, it has a severe drawback when considering systems with a fine surface discretization: due to q increasing with the number of points on the surface, the separated terms $\exp(qx)$ and $\exp(-qy)$ may overflow and underflow respectively. Tamaas will warn if that is the case, and users have two options to remedy the situation:

- Change the integration method by calling `setIntegrationMethod` with the desired `integration_method` on the `Model` object you use in the computation.
- Compile Tamaas with the option `real_type='long double'`. To make manipulation of numpy arrays easier, a `dtype` is provided in the `tamaas` module which can be used to create numpy arrays compatible with Tamaas' floating point type (e.g. `x = np.linspace(0, 1, dtype=tamaas.dtype)`)

Both these options negatively affect the performance, and it is up to the user to select the optimal solution for their particular use case.

8.3 Computational methods & Citations

Tamaas uses specialized numerical methods to efficiently solve elastic and elastoplastic periodic contact problems. Using a boundary integral formulation and a half-space geometry for the former allow (a) the focus of computational power to the contact interface since the bulk response can be represented exactly, (b) the use of the fast-Fourier transform for the computation of convolution integrals. In conjunction with a boundary integral formulation of the bulk state equations, a conjugate gradient approach is used to solve the contact problem.

Note: The above methods are state-of-the-art in the domain of rough surface contact. Below are selected publications detailing the methods used in elastic contact with and without adhesion:

- Boundary integral formulation:
 - Stanley and Kato (*J. of Tribology*, 1997)
- Conjugate Gradient:
 - Polonsky and Keer (*Wear*, 1999)
 - Rey, Anciaux and Molinari (*Computational Mechanics*, 2017)
- Frictional contact:
 - Condat (*J. of Optimization Theory and Applications*, 2012)

¹ L. Frérot, “Bridging scales in wear modeling with volume integral methods for elastic-plastic contact,” École Polytechnique Fédérale de Lausanne, 2020 (Section 2.3.2). doi:10.5075/epfl-thesis-7640.

For elastic-plastic contact, Tamaas uses a similar approach by implementing a *volume* integral formulation of the bulk equilibrium equations. Thanks to kernel expressions that are directly formulated in the Fourier domain, the method reduces the algorithmic complexity, memory requirements and sampling errors compared to traditional volume integral methods (Fr  rot, Bonnet, Anciaux and Molinari, [Computer Methods in Applied Mechanics and Engineering](#), 2019, [arxiv:1811.11558](#)). The figure below shows a comparison of run times for an elasticity problem (only a single solve step) between Tamaas and [Akantu](#), a high-performance FEM code using the direct solver [MUMPS](#).

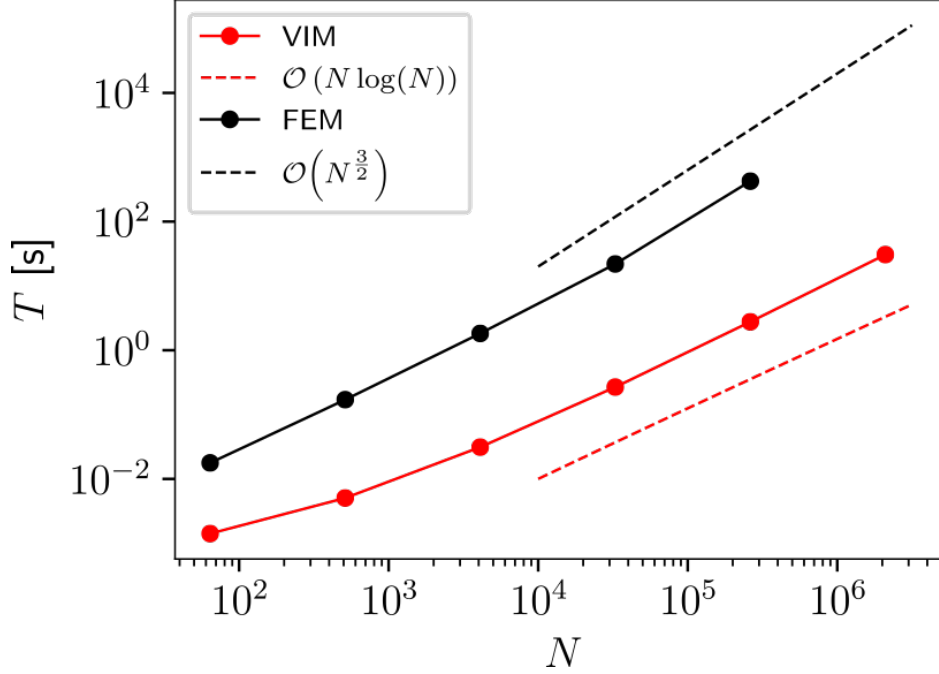


Fig. 1: Comparison of run times between the volume integral implementation (with cutoff integration) of Tamaas and an FEM solve step with a Cholesky factorization performed by Akantu+MUMPS. N is the total number of points.

Further discussion about the elastic-plastic solver implemented in Tamaas can be found in Fr  rot, Bonnet, Anciaux and Molinari, ([Computer Methods in Applied Mechanics and Engineering](#), 2019, [arxiv:1811.11558](#)).

FREQUENTLY ASKED QUESTIONS

9.1 Importing `tamaas` module gives a circular import error on Windows

(Similar to [this issue](#)) Installing with `pip install tamaas` does not work on Windows, as binary distributions are only built for Linux. To use Tamaas in Windows, use the Windows subsystem for Linux, then use `pip` to *install Tamaas*, or use the *provided Docker images*.

9.2 What are the units in Tamaas?

All quantities in Tamaas are unitless. To choose a consistent set of units, see *Units in Tamaas*.

9.3 Do contact solvers solve for a total force or an average pressure?

Solvers consider that the argument to the `solve()` method is an average apparent pressure (or average gap in some cases), i.e. the total force divided by the total system area (in the reference configuration). This means that doubling the system size and solving for the same argument to the solver *increases* the total load by a factor 4 (for a 2D surface).

9.4 `scons dev` fails to install Tamaas with externally-managed-environment error

Pip now refuses to install a package outside of virtual environments (see [PEP 668](#)), even with the `--user` flag on. This encourages the use of virtual environments, either with `venv` or `virtualenv`, which works nicely with `scons dev`.

However, if one **really** needs an editable installation outside of a virtual environment, the `PIPFLAGS` option can be used to circumvent pip's protections (not recommended):

```
scons dev PIPFLAGS="--user --break-system-packages"
```

Warning: If misused this can break your system's Python packages. Use virtual environments instead!

API REFERENCE

10.1 Python API

10.1.1 Tamaas root module

A high-performance library for periodic rough surface contact

See `__author__`, `__license__`, `__copyright__` for extra information about Tamaas.

- User documentation: <https://tamaas.readthedocs.io>
- Bug Tracker: <https://gitlab.com/tamaas/tamaas/-/issues>
- Source Code: <https://gitlab.com/tamaas/tamaas>

10.1.2 Tamaas C++ bindings

Compiled component of Tamaas

class `tamaas._tamaas.AdhesionFunctional`

Bases: *Functional*

__init__ (**args, **kwargs*)

computeF (*self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>*) → float

Compute functional value

computeGradF (*self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>*) → None

Compute functional gradient

property parameters

Parameters dictionary

setParameters (*self: tamaas._tamaas.AdhesionFunctional, arg0: Dict[str, float]*) → None

class `tamaas._tamaas.AndersonMixing`

Bases: *EPICSolver*

Fixed-point scheme with fixed memory size and Anderson mixing update to help and accelerate convergence. See doi:10.1006/jcph.1996.0059 for reference.

```
__init__ (self: tamaas._tamaas.AndersonMixing, contact_solver: tamaas._tamaas.ContactSolver,
          elasto_plastic_solver: tamaas._tamaas.EPSolver, tolerance: float = 1e-10, memory: int = 5) →
    None

acceleratedSolve (self: tamaas._tamaas.EPICSolver, normal_pressure: float) → float
    Solves the EP contact with an accelerated fixed-point scheme. May not converge!

property max_iter

property model

property relaxation

solve (self: tamaas._tamaas.EPICSolver, normal_pressure: float) → float
    Solves the EP contact with a relaxed fixed-point scheme. Adjust the relaxation parameter to help convergence.

property tolerance

exception tamaas._tamaas.AssertionError
    Bases: AssertionError

__init__ (*args, **kwargs)

args

with_traceback()
    Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class tamaas._tamaas.BEEngine
    Bases: pybind11_object

__init__ (*args, **kwargs)

getModel (self: tamaas._tamaas.BEEngine) → tamaas::Model

property model

registerDirichlet (self: tamaas._tamaas.BEEngine) → None

registerNeumann (self: tamaas._tamaas.BEEngine) → None

solveDirichlet (self: tamaas._tamaas.BEEngine, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) →
    None

solveNeumann (self: tamaas._tamaas.BEEngine, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) →
    None

class tamaas._tamaas.BeckTeboulle
    Bases: ContactSolver

__init__ (self: tamaas._tamaas.BeckTeboulle, model: tamaas._tamaas.Model, surface: GridBaseWrap<T>,
          tolerance: float, mu: float) → None

addFunctionalTerm (self: tamaas._tamaas.ContactSolver, arg0: tamaas._tamaas.Functional) → None
    Add a term to the contact functional to minimize

computeCost (self: tamaas._tamaas.BeckTeboulle, arg0: bool) → float
```

property dump_freq

Frequency of displaying solver info

property functional

property max_iter

Maximum number of iterations

property model

setDumpFrequency (*self*: [tamaas._tamaas.ContactSolver](#), *dump_freq*: *int*) → None

setMaxIterations (*self*: [tamaas._tamaas.ContactSolver](#), *max_iter*: *int*) → None

solve (*self*: [tamaas._tamaas.BeckTeboulle](#), *p0*: *GridBaseWrap<T>*) → float

property surface

property tolerance

Solver tolerance

class [tamaas._tamaas.Cluster1D](#)

Bases: [pybind11_object](#)

__init__ (*self*: [tamaas._tamaas.Cluster1D](#)) → None

property area

Area of cluster

property bounding_box

Compute the bounding box of a cluster

property extent

Compute the extents of a cluster

getArea (*self*: [tamaas._tamaas.Cluster1D](#)) → int

getPerimeter (*self*: [tamaas._tamaas.Cluster1D](#)) → int

getPoints (*self*: [tamaas._tamaas.Cluster1D](#)) → List[List[int[1]]]

property perimeter

Get perimeter of cluster

property points

Get list of points of cluster

class [tamaas._tamaas.Cluster2D](#)

Bases: [pybind11_object](#)

__init__ (*self*: [tamaas._tamaas.Cluster2D](#)) → None

property area

Area of cluster

property bounding_box

Compute the bounding box of a cluster

property extent

Compute the extents of a cluster

getArea (*self*: [tamaas._tamaas.Cluster2D](#)) → int

getPerimeter (*self*: [tamaas._tamaas.Cluster2D](#)) → int

getPoints (*self*: [tamaas._tamaas.Cluster2D](#)) → List[List[int[2]]]

property perimeter

Get perimeter of cluster

property points

Get list of points of cluster

class [tamaas._tamaas.Cluster3D](#)

Bases: [pybind11_object](#)

__init__ (*self*: [tamaas._tamaas.Cluster3D](#)) → None

property area

Area of cluster

property bounding_box

Compute the bounding box of a cluster

property extent

Compute the extents of a cluster

getArea (*self*: [tamaas._tamaas.Cluster3D](#)) → int

getPerimeter (*self*: [tamaas._tamaas.Cluster3D](#)) → int

getPoints (*self*: [tamaas._tamaas.Cluster3D](#)) → List[List[int[3]]]

property perimeter

Get perimeter of cluster

property points

Get list of points of cluster

class [tamaas._tamaas.Condat](#)

Bases: [ContactSolver](#)

Main solver for frictional contact problems. It has no restraint on the material properties or friction coefficient values, but solves an associated version of the Coulomb friction law, which differs from the traditional Coulomb friction in that the normal and tangential slip components are coupled.

__init__ (*self*: [tamaas._tamaas.Condat](#), *model*: [tamaas._tamaas.Model](#), *surface*: [GridBaseWrap<T>](#), *tolerance*: float, *mu*: float) → None

addFunctionalTerm (*self*: [tamaas._tamaas.ContactSolver](#), *arg0*: [tamaas._tamaas.Functional](#)) → None

Add a term to the contact functional to minimize

computeCost (*self*: [tamaas._tamaas.Condat](#), *arg0*: bool) → float

property dump_freq

Frequency of displaying solver info

property functional

property max_iter

Maximum number of iterations

property model

setDumpFrequency (*self*: [tamaas._tamaas.ContactSolver](#), *dump_freq*: *int*) → None

setMaxIterations (*self*: [tamaas._tamaas.ContactSolver](#), *max_iter*: *int*) → None

solve (*self*: [tamaas._tamaas.Condat](#), *p0*: [GridBaseWrap<T>](#), *grad_step*: *float* = 0.9) → float

property surface

property tolerance

Solver tolerance

class [tamaas._tamaas.ContactSolver](#)

Bases: [pybind11_object](#)

__init__ (*self*: [tamaas._tamaas.ContactSolver](#), *arg0*: [tamaas._tamaas.Model](#), *arg1*: [GridBaseWrap<T>](#), *arg2*: *float*) → None

addFunctionalTerm (*self*: [tamaas._tamaas.ContactSolver](#), *arg0*: [tamaas._tamaas.Functional](#)) → None

Add a term to the contact functional to minimize

property dump_freq

Frequency of displaying solver info

property functional

property max_iter

Maximum number of iterations

property model

setDumpFrequency (*self*: [tamaas._tamaas.ContactSolver](#), *dump_freq*: *int*) → None

setMaxIterations (*self*: [tamaas._tamaas.ContactSolver](#), *max_iter*: *int*) → None

solve (**args*, ***kwargs*)

Overloaded function.

1. `solve(self: tamaas.\_tamaas.ContactSolver, target_force: std::vector<double>, std::allocator<double> >)`
→ float

Solve the contact for a mean traction/gap vector

2. `solve(self: tamaas.\_tamaas.ContactSolver, target_normal_pressure: float)` → float

Solve the contact for a mean normal pressure/gap

property surface

property tolerance

Solver tolerance

class tamaas._tamaas.EPICSolver

Bases: pybind11_object

Main solver class for elastic-plastic contact problems

__init__ (*self*: tamaas._tamaas.EPICSolver, *contact_solver*: tamaas._tamaas.ContactSolver, *elasto_plastic_solver*: tamaas._tamaas.EPSolver, *tolerance*: float = 1e-10, *relaxation*: float = 0.3) → None

acceleratedSolve (*self*: tamaas._tamaas.EPICSolver, *normal_pressure*: float) → float

Solves the EP contact with an accelerated fixed-point scheme. May not converge!

property max_iter

property model

property relaxation

solve (*self*: tamaas._tamaas.EPICSolver, *normal_pressure*: float) → float

Solves the EP contact with a relaxed fixed-point scheme. Adjust the relaxation parameter to help convergence.

property tolerance

class tamaas._tamaas.EPSolver

Bases: pybind11_object

Mother class for nonlinear plasticity solvers

__init__ (*self*: tamaas._tamaas.EPSolver, *residual*: tamaas._tamaas.Residual) → None

beforeSolve (*self*: tamaas._tamaas.EPSolver) → None

getResidual (*self*: tamaas._tamaas.EPSolver) → *tamaas._tamaas.Residual*

getStrainIncrement (*self*: tamaas._tamaas.EPSolver) → GridBaseWrap<T>

setToleranceManager (*self*: tamaas._tamaas.EPSolver, *arg0*: tamaas._tamaas._tolerance_manager) → None

solve (*self*: tamaas._tamaas.EPSolver) → None

property tolerance

updateState (*self*: tamaas._tamaas.EPSolver) → None

class tamaas._tamaas.ElasticFunctionalGap

Bases: *Functional*

__init__ (*self*: tamaas._tamaas.ElasticFunctionalGap, *arg0*: tamaas::IntegralOperator, *arg1*: GridBaseWrap<T>) → None

computeF (*self*: tamaas._tamaas.Functional, *arg0*: GridBaseWrap<T>, *arg1*: GridBaseWrap<T>) → float

Compute functional value

computeGradF (*self*: tamaas._tamaas.Functional, *arg0*: GridBaseWrap<T>, *arg1*: GridBaseWrap<T>) → None

Compute functional gradient

class tamaas._tamaas.ElasticFunctionalPressure

Bases: *Functional*

```

__init__ (self: tamaas._tamaas.ElasticFunctionalPressure, arg0: tamaas::IntegralOperator, arg1:
    GridBaseWrap<T>) → None

computeF (self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) → float
    Compute functional value

computeGradF (self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) →
    None
    Compute functional gradient

class tamaas._tamaas.ExponentialAdhesionFunctional
    Bases: AdhesionFunctional
    Potential of the form  $F = -\gamma \cdot \exp(-g/\rho)$ 

    __init__ (self: tamaas._tamaas.ExponentialAdhesionFunctional, surface: GridBaseWrap<T>) → None

    computeF (self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) → float
        Compute functional value

    computeGradF (self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) →
        None
        Compute functional gradient

    property parameters
        Parameters dictionary

    setParameters (self: tamaas._tamaas.AdhesionFunctional, arg0: Dict[str, float]) → None

class tamaas._tamaas.FieldContainer
    Bases: pybind11_object

    __init__ (self: tamaas._tamaas.FieldContainer) → None

    getField (self: tamaas::Model, field_name: str) → GridVariant

    getFields (self: tamaas::Model) → List[str]
        Return fields list

    registerField (self: tamaas::Model, field_name: str, field: numpy.ndarray[numpy.float64]) → None

class tamaas._tamaas.Filter1D
    Bases: pybind11_object
    Mother class for Fourier filter objects

    __init__ (self: tamaas._tamaas.Filter1D) → None

    computeFilter (self: tamaas._tamaas.Filter1D, arg0: GridWrap<T, dim>) → None
        Compute the Fourier coefficient of the surface

class tamaas._tamaas.Filter2D
    Bases: pybind11_object
    Mother class for Fourier filter objects

    __init__ (self: tamaas._tamaas.Filter2D) → None

    computeFilter (self: tamaas._tamaas.Filter2D, arg0: GridWrap<T, dim>) → None
        Compute the Fourier coefficient of the surface

```

exception tamaas._tamaas.FloatingPointError

Bases: *FloatingPointError*

__init__ (*args, **kwargs)

args

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class tamaas._tamaas.FloodFill

Bases: pybind11_object

__init__ (*args, **kwargs)

static getClusters (contact: GridWrap<T, dim>, diagonal: bool) → List[tamaas._tamaas.Cluster2D]

Return a list of clusters from boolean map

static getSegments (contact: GridWrap<T, dim>) → List[tamaas._tamaas.Cluster1D]

Return a list of segments from boolean map

static getVolumes (map: GridWrap<T, dim>, diagonal: bool) → List[tamaas._tamaas.Cluster3D]

Return a list of volume clusters

class tamaas._tamaas.Functional

Bases: pybind11_object

__init__ (self: tamaas._tamaas.Functional) → None

computeF (self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) → float

Compute functional value

computeGradF (self: tamaas._tamaas.Functional, arg0: GridBaseWrap<T>, arg1: GridBaseWrap<T>) →

None

Compute functional gradient

class tamaas._tamaas.IntegralOperator

Bases: pybind11_object

__init__ (self: tamaas._tamaas.IntegralOperator, arg0: tamaas._tamaas.Model) → None

apply (self: tamaas._tamaas.IntegralOperator, arg0: numpy.ndarray[numpy.float64], arg1: numpy.ndarray[numpy.float64]) → None

dirac = <kind.dirac: 2>

dirichlet = <kind.dirichlet: 1>

getKind (self: tamaas._tamaas.IntegralOperator) → tamaas._tamaas.IntegralOperator.kind

getModel (self: tamaas._tamaas.IntegralOperator) → tamaas._tamaas.Model

getType (self: tamaas._tamaas.IntegralOperator) → tamaas._tamaas.model_type

property kind

matvec (self: tamaas._tamaas.IntegralOperator, arg0: numpy.ndarray[numpy.float64]) → GridBaseWrap<T>

property model

```

    neumann = <kind.neumann: 0>

    property shape

    property type

    updateFromModel (self: tamaas._tamaas.IntegralOperator) → None
        Resets internal persistent variables from the model

class tamaas._tamaas.Isopowerlaw1D
    Bases: Filter1D

    Isotropic powerlaw spectrum with a rolloff plateau

    __init__ (self: tamaas._tamaas.Isopowerlaw1D) → None

    alpha (self: tamaas._tamaas.Isopowerlaw1D) → float
        Nayak's bandwidth parameter

    computeFilter (self: tamaas._tamaas.Filter1D, arg0: GridWrap<T, dim>) → None
        Compute the Fourier coefficient of the surface

    elasticEnergy (self: tamaas._tamaas.Isopowerlaw1D) → float
        Computes full contact energy (adimensional)

    property hurst
        Hurst exponent

    moments (self: tamaas._tamaas.Isopowerlaw1D) → List[float]
        Theoretical first 3 moments of spectrum

    property q0
        Long wavelength cutoff

    property q1
        Rolloff wavelength

    property q2
        Short wavelength cutoff

    radialPSDMoment (self: tamaas._tamaas.Isopowerlaw1D, arg0: float) → float
        Computes  $\int k^q \phi(k) k dk$  from 0 to  $\infty$ 

    rmsHeights (self: tamaas._tamaas.Isopowerlaw1D) → float
        Theoretical RMS of heights

    rmsSlopes (self: tamaas._tamaas.Isopowerlaw1D) → float
        Theoretical RMS of slopes

class tamaas._tamaas.Isopowerlaw2D
    Bases: Filter2D

    Isotropic powerlaw spectrum with a rolloff plateau

    __init__ (self: tamaas._tamaas.Isopowerlaw2D) → None

    alpha (self: tamaas._tamaas.Isopowerlaw2D) → float
        Nayak's bandwidth parameter

```

computeFilter (*self*: *tamaas._tamaas.Filter2D*, *arg0*: *GridWrap<T, dim>*) → None
Compute the Fourier coefficient of the surface

elasticEnergy (*self*: *tamaas._tamaas.Isopowerlaw2D*) → float
Computes full contact energy (adimensional)

property hurst
Hurst exponent

moments (*self*: *tamaas._tamaas.Isopowerlaw2D*) → List[float]
Theoretical first 3 moments of spectrum

property q0
Long wavelength cutoff

property q1
Rolloff wavelength

property q2
Short wavelength cutoff

radialPSDMoment (*self*: *tamaas._tamaas.Isopowerlaw2D*, *arg0*: *float*) → float
Computes $\int k^q \phi(k) k dk$ from 0 to ∞

rmsHeights (*self*: *tamaas._tamaas.Isopowerlaw2D*) → float
Theoretical RMS of heights

rmsSlopes (*self*: *tamaas._tamaas.Isopowerlaw2D*) → float
Theoretical RMS of slopes

class *tamaas._tamaas.Kato*
Bases: *ContactSolver*

__init__ (*self*: *tamaas._tamaas.Kato*, *model*: *tamaas._tamaas.Model*, *surface*: *GridBaseWrap<T>*, *tolerance*: *float*, *mu*: *float*) → None

addFunctionalTerm (*self*: *tamaas._tamaas.ContactSolver*, *arg0*: *tamaas._tamaas.Functional*) → None
Add a term to the contact functional to minimize

computeCost (*self*: *tamaas._tamaas.Kato*, *use_tresca*: *bool = False*) → float

property dump_freq
Frequency of displaying solver info

property functional

property max_iter
Maximum number of iterations

property model

setDumpFrequency (*self*: *tamaas._tamaas.ContactSolver*, *dump_freq*: *int*) → None

setMaxIterations (*self*: *tamaas._tamaas.ContactSolver*, *max_iter*: *int*) → None

solve (*self*: *tamaas._tamaas.Kato*, *p0*: *GridBaseWrap<T>*, *proj_iter*: *int = 50*) → float

solveRegularized (*self*: *tamaas._tamaas.Kato*, *p0*: *GridBaseWrap<T>*, *r*: *float = 0.01*) → float

solveRelaxed (*self*: *tamaas._tamaas.Kato*, *g0*: *GridBaseWrap<T>*) → float

property surface

property tolerance

Solver tolerance

class *tamaas._tamaas.KatoSaturated*

Bases: *PolonskyKeerRey*

Solver for pseudo-plasticity problems where the normal pressure is constrained above by a saturation pressure “pmax”

__init__ (*self*: *tamaas._tamaas.KatoSaturated*, *model*: *tamaas._tamaas.Model*, *surface*: *GridBaseWrap<T>*, *tolerance*: float, *pmax*: float) → None

addFunctionalTerm (*self*: *tamaas._tamaas.ContactSolver*, *arg0*: *tamaas._tamaas.Functional*) → None

Add a term to the contact functional to minimize

computeError (*self*: *tamaas._tamaas.PolonskyKeerRey*) → float

property dump_freq

Frequency of displaying solver info

property functional

gap = <type.gap: 0>

property max_iter

Maximum number of iterations

property model

property pmax

Saturation normal pressure

pressure = <type.pressure: 1>

setDumpFrequency (*self*: *tamaas._tamaas.ContactSolver*, *dump_freq*: int) → None

setIntegralOperator (*self*: *tamaas._tamaas.PolonskyKeerRey*, *arg0*: str) → None

setMaxIterations (*self*: *tamaas._tamaas.ContactSolver*, *max_iter*: int) → None

solve (*args, **kwargs)

Overloaded function.

1. **solve**(*self*: *tamaas._tamaas.ContactSolver*, *target_force*: std::vector<double, std::allocator<double> >) -> float

Solve the contact for a mean traction/gap vector

2. **solve**(*self*: *tamaas._tamaas.ContactSolver*, *target_normal_pressure*: float) -> float

Solve the contact for a mean normal pressure/gap

property surface

property tolerance

Solver tolerance

class typeBases: `pybind11_object`

Members:

`gap``pressure``__init__` (*self*: `tamaas._tamaas.PolonskyKeerRey.type`, *value*: `int`) → `None``gap` = `<type.gap: 0>``property name``pressure` = `<type.pressure: 1>``property value`**class** `tamaas._tamaas.LogLevel`Bases: `pybind11_object`

Members:

`debug``info``warning``error``__init__` (*self*: `tamaas._tamaas.LogLevel`, *value*: `int`) → `None``debug` = `<LogLevel.debug: 0>``error` = `<LogLevel.error: 3>``info` = `<LogLevel.info: 1>``property name``property value``warning` = `<LogLevel.warning: 2>`**class** `tamaas._tamaas.Logger`Bases: `pybind11_object``__init__` (*self*: `tamaas._tamaas.Logger`) → `None``get` (*self*: `tamaas._tamaas.Logger`, *arg0*: `tamaas._tamaas.LogLevel`) → `tamaas._tamaas.Logger`

Get a logger object for a log level

class `tamaas._tamaas.MaugisAdhesionFunctional`Bases: `AdhesionFunctional`Cohesive zone potential $F = H(g - \rho) \cdot \gamma / \rho$ `__init__` (*self*: `tamaas._tamaas.MaugisAdhesionFunctional`, *surface*: `GridBaseWrap<T>`) → `None``computeF` (*self*: `tamaas._tamaas.Functional`, *arg0*: `GridBaseWrap<T>`, *arg1*: `GridBaseWrap<T>`) → `float`

Compute functional value

computeGradF (*self*: *tamaas._tamaas.Functional*, *arg0*: *GridBaseWrap<T>*, *arg1*: *GridBaseWrap<T>*) → None

Compute functional gradient

property parameters

Parameters dictionary

setParameters (*self*: *tamaas._tamaas.AdhesionFunctional*, *arg0*: *Dict[str, float]*) → None

class *tamaas._tamaas.Model*

Bases: *FieldContainer*

property E

Young's modulus

property E_star

Contact (Hertz) modulus

__init__ (*self*: *tamaas._tamaas.Model*, *arg0*: *tamaas._tamaas.model_type*, *arg1*: *List[float]*, *arg2*: *List[int]*) → None

Create a new model of a given type, physical size and *global* discretization.

Parameters

- **model_type** – the type of desired model
- **system_size** – the physical size of the domain in each direction
- **global_discretization** – number of points in each direction

addDumper (*self*: *tamaas._tamaas.Model*, *dumper*: *tamaas::ModelDumper*) → None

Register a dumper

applyElasticity (*self*: *tamaas._tamaas.Model*, *arg0*: *numpy.ndarray[numpy.float64]*, *arg1*: *numpy.ndarray[numpy.float64]*) → None

Apply Hooke's law

property be_engine

Boundary element engine

property boundary_fields

property boundary_shape

Number of points on boundary

property boundary_system_size

Physical size of surface

property displacement

Displacement field

dump (*self*: *tamaas._tamaas.Model*) → None

Write model data to registered dumpers

getBEEngine (*self*: *tamaas._tamaas.Model*) → *tamaas._tamaas.BEEngine*

getBoundaryDiscretization (*self*: *tamaas._tamaas.Model*) → *List[int]*

getBoundarySystemSize (*self*: *tamaas._tamaas.Model*) → *List[float]*

getDiscretization (*self*: `tamaas._tamaas.Model`) → List[int]

getDisplacement (*self*: `tamaas._tamaas.Model`) → GridBaseWrap<T>

getField (*self*: `tamaas::Model`, *field_name*: str) → GridVariant

getFields (*self*: `tamaas::Model`) → List[str]
Return fields list

getHertzModulus (*self*: `tamaas._tamaas.Model`) → float

getIntegralOperator (*self*: `tamaas._tamaas.Model`, *operator_name*: str) → `tamaas::IntegralOperator`

getPoissonRatio (*self*: `tamaas._tamaas.Model`) → float

getShearModulus (*self*: `tamaas._tamaas.Model`) → float

getSystemSize (*self*: `tamaas._tamaas.Model`) → List[float]

getTraction (*self*: `tamaas._tamaas.Model`) → GridBaseWrap<T>

getYoungModulus (*self*: `tamaas._tamaas.Model`) → float

property global_shape
Global discretization (in MPI environnement)

property mu
Shear modulus

property nu
Poisson's ratio

property operators
Returns a dict-like object allowing access to the model's integral operators

registerField (*self*: `tamaas::Model`, *field_name*: str, *field*: `numpy.ndarray[numpy.float64]`) → None

setElasticity (*self*: `tamaas._tamaas.Model`, *E*: float, *nu*: float) → None

setIntegrationMethod (*self*: `tamaas._tamaas.Model`, *arg0*: `tamaas::integration_method`, *arg1*: float) → None

property shape
Discretization (local in MPI environment)

solveDirichlet (*self*: `tamaas._tamaas.Model`) → None
Solve surface displacements -> tractions

solveNeumann (*self*: `tamaas._tamaas.Model`) → None
Solve surface tractions -> displacements

property system_size
Size of physical domain

property traction
Surface traction field

property type

```

class tamaas._tamaas.ModelDumper
    Bases: pybind11_object
    __init__ (self: tamaas._tamaas.ModelDumper) → None

    dump (self: tamaas._tamaas.ModelDumper, model: tamaas._tamaas.Model) → None
        Dump model

class tamaas._tamaas.ModelFactory
    Bases: pybind11_object
    __init__ (*args, **kwargs)

    static createModel (*args, **kwargs)
        Overloaded function.

        1. createModel(model_type: tamaas._tamaas.model_type, system_size: List[float], global_discretization:
            List[int]) -> tamaas._tamaas.Model

            Create a new model of a given type, physical size and global discretization.

            Parameters
            • model_type – the type of desired model
            • system_size – the physical size of the domain in each direction
            • global_discretization – number of points in each direction

        2. createModel(model: tamaas._tamaas.Model) -> tamaas._tamaas.Model

            Create a deep copy of a model.

    static createResidual (model: tamaas._tamaas.Model, sigma_y: float, hardening: float = 0.0) →
        tamaas::Residual

        Create an isotropic linear hardening residual. :param model: the model on which to define the residual :param
        sigma_y: the (von Mises) yield stress :param hardening: the hardening modulus

    static registerHookeField (model: tamaas._tamaas.Model, name: str) → None
        Register a HookeField operator

    static registerNonPeriodic (model: tamaas._tamaas.Model, name: str) → None
        Register non-periodic Boussinesq operator

    static registerVolumeOperators (model: tamaas._tamaas.Model) → None
        Register Boussinesq and Mindlin operators to model.

    static setIntegrationMethod (operator: tamaas::IntegralOperator, method:
        tamaas::integration_method, cutoff: float) → None

        Set the integration method (linear or cutoff) for a volume integral operator

exception tamaas._tamaas.ModelTypeError
    Bases: TypeError
    __init__ (*args, **kwargs)

    args

    with_traceback ()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```

exception tamaas._tamaas.NotImplementedError

Bases: *NotImplementedError*

__init__ (*args, **kwargs)

args

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class tamaas._tamaas.PolonskyKeerRey

Bases: *ContactSolver*

Main solver class for normal elastic contact problems. Its functional can be customized to add an adhesion term, and its primal variable can be set to either the gap or the pressure.

__init__ (self: tamaas._tamaas.PolonskyKeerRey, model: tamaas._tamaas.Model, surface: GridBaseWrap<T>, tolerance: float, primal_type: tamaas._tamaas.PolonskyKeerRey.type = <type.pressure: 1>, constraint_type: tamaas._tamaas.PolonskyKeerRey.type = <type.pressure: 1>) → None

addFunctionalTerm (self: tamaas._tamaas.ContactSolver, arg0: tamaas._tamaas.Functional) → None

Add a term to the contact functional to minimize

computeError (self: tamaas._tamaas.PolonskyKeerRey) → float

property dump_freq

Frequency of displaying solver info

property functional

gap = <type.gap: 0>

property max_iter

Maximum number of iterations

property model

pressure = <type.pressure: 1>

setDumpFrequency (self: tamaas._tamaas.ContactSolver, dump_freq: int) → None

setIntegralOperator (self: tamaas._tamaas.PolonskyKeerRey, arg0: str) → None

setMaxIterations (self: tamaas._tamaas.ContactSolver, max_iter: int) → None

solve (*args, **kwargs)

Overloaded function.

1. solve(self: tamaas._tamaas.ContactSolver, target_force: std::vector<double, std::allocator<double> >) → float

Solve the contact for a mean traction/gap vector

2. solve(self: tamaas._tamaas.ContactSolver, target_normal_pressure: float) → float

Solve the contact for a mean normal pressure/gap

property surface

property tolerance

Solver tolerance

class typeBases: `pybind11_object`

Members:

`gap``pressure``__init__` (*self*: `tamaas._tamaas.PolonskyKeerRey.type`, *value*: `int`) → `None``gap` = `<type.gap: 0>`**property name**`pressure` = `<type.pressure: 1>`**property value****class** `tamaas._tamaas.PolonskyKeerTan`Bases: `ContactSolver``__init__` (*self*: `tamaas._tamaas.PolonskyKeerTan`, *model*: `tamaas._tamaas.Model`, *surface*: `GridBaseWrap<T>`, *tolerance*: `float`, *mu*: `float`) → `None``addFunctionalTerm` (*self*: `tamaas._tamaas.ContactSolver`, *arg0*: `tamaas._tamaas.Functional`) → `None`
Add a term to the contact functional to minimize`computeCost` (*self*: `tamaas._tamaas.PolonskyKeerTan`, *use_tresca*: `bool = False`) → `float`**property dump_freq**

Frequency of displaying solver info

property functional**property max_iter**

Maximum number of iterations

property model`setDumpFrequency` (*self*: `tamaas._tamaas.ContactSolver`, *dump_freq*: `int`) → `None``setMaxIterations` (*self*: `tamaas._tamaas.ContactSolver`, *max\_iter*: `int`) → `None``solve` (*self*: `tamaas._tamaas.PolonskyKeerTan`, *p0*: `GridBaseWrap<T>`) → `float``solveTresca` (*self*: `tamaas._tamaas.PolonskyKeerTan`, *p0*: `GridBaseWrap<T>`) → `float`**property surface****property tolerance**

Solver tolerance

class `tamaas._tamaas.RegularizedPowerlaw1D`Bases: `Filter1D`

Isotropic regularized powerlaw with a plateau extending to the size of the system

__init__ (*self*: tamaas._tamaas.RegularizedPowerlaw1D) → None

computeFilter (*self*: tamaas._tamaas.Filter1D, *arg0*: GridWrap<T, dim>) → None

Compute the Fourier coefficient of the surface

property hurst

Hurst exponent

property q1

Rolloff wavelength

property q2

Short wavelength cutoff

class tamaas._tamaas.RegularizedPowerlaw2D

Bases: *Filter2D*

Isotropic regularized powerlaw with a plateau extending to the size of the system

__init__ (*self*: tamaas._tamaas.RegularizedPowerlaw2D) → None

computeFilter (*self*: tamaas._tamaas.Filter2D, *arg0*: GridWrap<T, dim>) → None

Compute the Fourier coefficient of the surface

property hurst

Hurst exponent

property q1

Rolloff wavelength

property q2

Short wavelength cutoff

class tamaas._tamaas.Residual

Bases: *pybind11_object*

__init__ (*self*: tamaas._tamaas.Residual, *model*: tamaas._tamaas.Model, *material*: tamaas::Material) → None

Create a residual object with a material. Defines the following residual equation:

$$\varepsilon - \nabla N[\sigma(\varepsilon)] - \nabla M[t] = 0$$

Where $\sigma(\varepsilon)$ is the *eigenstress* associated with the constitutive law.

Parameters

- **model** – the model on which to define the residual
- **material** – material object which defines a constitutive law

applyTangent (*self*: tamaas._tamaas.Residual, *arg0*: GridBaseWrap<T>, *arg1*: GridBaseWrap<T>, *arg2*: GridBaseWrap<T>) → None

computeResidual (*self*: tamaas._tamaas.Residual, *arg0*: GridBaseWrap<T>) → None

computeResidualDisplacement (*self*: tamaas._tamaas.Residual, *arg0*: GridBaseWrap<T>) → None

getStress (*self*: tamaas._tamaas.Residual) → GridWrap<T, dim>

getVector (*self*: tamaas._tamaas.Residual) → GridBaseWrap<T>

property material

property model

setIntegrationMethod (*self*: *tamaas._tamaas.Residual*, *arg0*: *tamaas::integration_method*, *arg1*: *float*)
→ None

updateState (*self*: *tamaas._tamaas.Residual*, *arg0*: *GridBaseWrap<T>*) → None

property vector

class *tamaas._tamaas.SquaredExponentialAdhesionFunctional*

Bases: *AdhesionFunctional*

Potential of the form $F = -\gamma \cdot \exp(-0.5 \cdot (g/\rho)^2)$

__init__ (*self*: *tamaas._tamaas.SquaredExponentialAdhesionFunctional*, *surface*: *GridBaseWrap<T>*) →
None

computeF (*self*: *tamaas._tamaas.Functional*, *arg0*: *GridBaseWrap<T>*, *arg1*: *GridBaseWrap<T>*) → float
Compute functional value

computeGradF (*self*: *tamaas._tamaas.Functional*, *arg0*: *GridBaseWrap<T>*, *arg1*: *GridBaseWrap<T>*) →
None
Compute functional gradient

property parameters

Parameters dictionary

setParameters (*self*: *tamaas._tamaas.AdhesionFunctional*, *arg0*: *Dict[str, float]*) → None

class *tamaas._tamaas.Statistics1D*

Bases: *pybind11_object*

__init__ (**args*, ***kwargs*)

static computeAutocorrelation (*arg0*: *GridWrap<T, dim>*) → *GridWrap<T, dim>*
Compute autocorrelation of surface

static computeFDRMSSlope (*arg0*: *GridWrap<T, dim>*) → float
Compute hrms' with finite differences

static computeMoments (*arg0*: *GridWrap<T, dim>*) → *List[float]*
Compute spectral moments

static computePowerSpectrum (*arg0*: *GridWrap<T, dim>*) → *GridWrap<T, dim>*
Compute PSD of surface

static computeRMSHeights (*arg0*: *GridWrap<T, dim>*) → float
Compute hrms

static computeSpectralRMSSlope (*arg0*: *GridWrap<T, dim>*) → float
Compute hrms' in Fourier space

static contact (*tractions*: *GridWrap<T, dim>*, *perimeter*: *int = 0*) → float
Compute the (corrected) contact area. Perimeter is the total contact perimeter in number of segments.

static graphArea (*zdisplacement*: *GridWrap<T, dim>*) → float
Compute area defined by function graph.

```
class tamaas._tamaas.Statistics2D
    Bases: pybind11_object
    __init__ (*args, **kwargs)

    static computeAutocorrelation (arg0: GridWrap<T, dim>) → GridWrap<T, dim>
        Compute autocorrelation of surface

    static computeFDRMSSlope (arg0: GridWrap<T, dim>) → float
        Compute hrms' with finite differences

    static computeMoments (arg0: GridWrap<T, dim>) → List[float]
        Compute spectral moments

    static computePowerSpectrum (arg0: GridWrap<T, dim>) → GridWrap<T, dim>
        Compute PSD of surface

    static computeRMSHeights (arg0: GridWrap<T, dim>) → float
        Compute hrms

    static computeSpectralRMSSlope (arg0: GridWrap<T, dim>) → float
        Compute hrms' in Fourier space

    static contact (tractions: GridWrap<T, dim>, perimeter: int = 0) → float
        Compute the (corrected) contact area. Perimeter is the total contact perimeter in number of segments.

    static graphArea (zdisplacement: GridWrap<T, dim>) → float
        Compute area defined by function graph.

class tamaas._tamaas.SurfaceGenerator1D
    Bases: pybind11_object
    __init__ (*args, **kwargs)

    buildSurface (self: tamaas._tamaas.SurfaceGenerator1D) → GridWrap<T, dim>
        Generate a surface and return a reference to it

    property random_seed
        Random generator seed

    setRandomSeed (self: tamaas._tamaas.SurfaceGenerator1D, arg0: int) → None

    setSizes (self: tamaas._tamaas.SurfaceGenerator1D, arg0: List[int[1]]) → None

    property shape
        Global shape of surfaces

class tamaas._tamaas.SurfaceGenerator2D
    Bases: pybind11_object
    __init__ (*args, **kwargs)

    buildSurface (self: tamaas._tamaas.SurfaceGenerator2D) → GridWrap<T, dim>
        Generate a surface and return a reference to it

    property random_seed
        Random generator seed

    setRandomSeed (self: tamaas._tamaas.SurfaceGenerator2D, arg0: int) → None
```

setSize (*self*: tamaas._tamaas.SurfaceGenerator2D, *arg0*: List[int[2]]) → None

property shape

Global shape of surfaces

class tamaas._tamaas.SurfaceGeneratorFilter1D

Bases: *SurfaceGenerator1D*

Generates a rough surface with Gaussian noise in the PSD

__init__ (**args*, ***kwargs*)

Overloaded function.

1. **__init__**(*self*: tamaas._tamaas.SurfaceGeneratorFilter1D) -> None

Default constructor

2. **__init__**(*self*: tamaas._tamaas.SurfaceGeneratorFilter1D, *arg0*: List[int[1]]) -> None

Initialize with global surface shape

buildSurface (*self*: tamaas._tamaas.SurfaceGenerator1D) → GridWrap<T, dim>

Generate a surface and return a reference to it

property random_seed

Random generator seed

setFilter (*self*: tamaas._tamaas.SurfaceGeneratorFilter1D, *filter*: tamaas._tamaas.Filter1D) → None

Set PSD filter

setRandomSeed (*self*: tamaas._tamaas.SurfaceGenerator1D, *arg0*: int) → None

setSize (*self*: tamaas._tamaas.SurfaceGenerator1D, *arg0*: List[int[1]]) → None

setSpectrum (*self*: tamaas._tamaas.SurfaceGeneratorFilter1D, *filter*: tamaas._tamaas.Filter1D) → None

Set PSD filter

property shape

Global shape of surfaces

property spectrum

Power spectrum object

class tamaas._tamaas.SurfaceGeneratorFilter2D

Bases: *SurfaceGenerator2D*

Generates a rough surface with Gaussian noise in the PSD

__init__ (**args*, ***kwargs*)

Overloaded function.

1. **__init__**(*self*: tamaas._tamaas.SurfaceGeneratorFilter2D) -> None

Default constructor

2. **__init__**(*self*: tamaas._tamaas.SurfaceGeneratorFilter2D, *arg0*: List[int[2]]) -> None

Initialize with global surface shape

buildSurface (*self*: tamaas._tamaas.SurfaceGenerator2D) → GridWrap<T, dim>

Generate a surface and return a reference to it

property random_seed

Random generator seed

setFilter (*self*: tamaas._tamaas.SurfaceGeneratorFilter2D, *filter*: tamaas._tamaas.Filter2D) → None

Set PSD filter

setRandomSeed (*self*: tamaas._tamaas.SurfaceGenerator2D, *arg0*: int) → None**setSize** (*self*: tamaas._tamaas.SurfaceGenerator2D, *arg0*: List[int[2]]) → None**setSpectrum** (*self*: tamaas._tamaas.SurfaceGeneratorFilter2D, *filter*: tamaas._tamaas.Filter2D) → None

Set PSD filter

property shape

Global shape of surfaces

property spectrum

Power spectrum object

class tamaas._tamaas.SurfaceGeneratorRandomPhase1DBases: *SurfaceGeneratorFilter1D*

Generates a rough surface with uniformly distributed phases and exact prescribed PSD

__init__ (*args, **kwargs)

Overloaded function.

1. **__init__**(*self*: tamaas._tamaas.SurfaceGeneratorRandomPhase1D) -> None

Default constructor

2. **__init__**(*self*: tamaas._tamaas.SurfaceGeneratorRandomPhase1D, *arg0*: List[int[1]]) -> None

Initialize with global surface shape

buildSurface (*self*: tamaas._tamaas.SurfaceGenerator1D) → GridWrap<T, dim>

Generate a surface and return a reference to it

property random_seed

Random generator seed

setFilter (*self*: tamaas._tamaas.SurfaceGeneratorFilter1D, *filter*: tamaas._tamaas.Filter1D) → None

Set PSD filter

setRandomSeed (*self*: tamaas._tamaas.SurfaceGenerator1D, *arg0*: int) → None**setSize** (*self*: tamaas._tamaas.SurfaceGenerator1D, *arg0*: List[int[1]]) → None**setSpectrum** (*self*: tamaas._tamaas.SurfaceGeneratorFilter1D, *filter*: tamaas._tamaas.Filter1D) → None

Set PSD filter

property shape

Global shape of surfaces

property spectrum

Power spectrum object

class tamaas._tamaas.SurfaceGeneratorRandomPhase2DBases: *SurfaceGeneratorFilter2D*

Generates a rough surface with uniformly distributed phases and exact prescribed PSD

__init__ (*args, **kwargs)

Overloaded function.

1. **__init__**(self: tamaas._tamaas.SurfaceGeneratorRandomPhase2D) -> None

Default constructor

2. **__init__**(self: tamaas._tamaas.SurfaceGeneratorRandomPhase2D, arg0: List[int[2]]) -> None

Initialize with global surface shape

buildSurface (self: tamaas._tamaas.SurfaceGenerator2D) → GridWrap<T, dim>

Generate a surface and return a reference to it

property random_seed

Random generator seed

setFilter (self: tamaas._tamaas.SurfaceGeneratorFilter2D, filter: tamaas._tamaas.Filter2D) → None

Set PSD filter

setRandomSeed (self: tamaas._tamaas.SurfaceGenerator2D, arg0: int) → None

setSize (self: tamaas._tamaas.SurfaceGenerator2D, arg0: List[int[2]]) → None

setSpectrum (self: tamaas._tamaas.SurfaceGeneratorFilter2D, filter: tamaas._tamaas.Filter2D) → None

Set PSD filter

property shape

Global shape of surfaces

property spectrum

Power spectrum object

class tamaas._tamaas.TamaasInfo

Bases: pybind11_object

__init__ (*args, **kwargs)

backend = 'cpp'

branch = ''

build_type = 'release'

commit = ''

diff = ''

has_mpi = False

remotes = ''

version = '2.7.0+5.g7767124'

tamaas._tamaas.**finalize** () → None

tamaas._tamaas.**get_log_level** () → *tamaas._tamaas.LogLevel*

tamaas._tamaas.**initialize** (num_threads: int = 0) → None

Initialize tamaas with desired number of threads. Automatically called upon import of the tamaas module, but can be manually called to set the desired number of threads.

class tamaas._tamaas.integration_method

Bases: pybind11_object

Integration method used for the computation of volumetric Fourier operators

Members:

linear : No approximation error, $O(N_1 \cdot N_2 \cdot N_3)$ time complexity, may cause float overflow/underflow

cutoff : Approximation, $O(\sqrt{N_1^2 + N_2^2} \cdot N_3^2)$ time complexity, no overflow/underflow risk

__init__ (self: tamaas._tamaas.integration_method, value: int) → None

cutoff = <integration_method.cutoff: 0>

linear = <integration_method.linear: 1>

property name

property value

class tamaas._tamaas.model_type

Bases: pybind11_object

Members:

basic_1d : Normal contact with 1D interface

basic_2d : Normal contact with 2D interface

surface_1d : Normal & tangential contact with 1D interface

surface_2d : Normal & tangential contact with 2D interface

volume_1d : Contact with volumetric representation and 1D interface

volume_2d : Contact with volumetric representation and 2D interface

__init__ (self: tamaas._tamaas.model_type, value: int) → None

basic_1d = <model_type.basic_1d: 0>

basic_2d = <model_type.basic_2d: 1>

property name

surface_1d = <model_type.surface_1d: 2>

surface_2d = <model_type.surface_2d: 3>

property value

volume_1d = <model_type.volume_1d: 4>

volume_2d = <model_type.volume_2d: 5>

tamaas._tamaas.set_log_level (arg0: tamaas._tamaas.LogLevel) → None

tamaas._tamaas.to_voigt (arg0: GridWrap<T, dim>) → GridWrap<T, dim>

Convert a 3D tensor field to voigt notation

class tamaas._tamaas._DFSANESolver

Bases: EPSolver

__init__ (*args, **kwargs)

Overloaded function.

1. **__init__**(self: tamaas._tamaas.DFSANESolver, residual: tamaas._tamaas.Residual) -> None
2. **__init__**(self: tamaas._tamaas.DFSANESolver, residual: tamaas._tamaas.Residual, model: tamaas._tamaas.Model) -> None

beforeSolve (self: tamaas._tamaas.EPSolver) → None

getResidual (self: tamaas._tamaas.EPSolver) → *tamaas._tamaas.Residual*

getStrainIncrement (self: tamaas._tamaas.EPSolver) → GridBaseWrap<T>

property max_iter

setToleranceManager (self: tamaas._tamaas.EPSolver, arg0: tamaas._tamaas._tolerance_manager) → None

solve (self: tamaas._tamaas.EPSolver) → None

property tolerance

updateState (self: tamaas._tamaas.EPSolver) → None

Module defining convenience MPI routines.

tamaas._tamaas.mpi.**gather** (arg0: GridWrap<T, dim>) → GridWrap<T, dim>

Gather 2D surfaces

tamaas._tamaas.mpi.**global_shape** (local_shape: List[int]) → List[int]

Gives the global shape of a 1D/2D local shape

tamaas._tamaas.mpi.**local_offset** (global_shape: List[int]) → int

Gives the local offset of a 1D/2D global shape

tamaas._tamaas.mpi.**local_shape** (global_shape: List[int]) → List[int]

Gives the local size of a 1D/2D global shape

tamaas._tamaas.mpi.**rank**() → int

Returns the rank of the local process

tamaas._tamaas.mpi.**scatter** (arg0: GridWrap<T, dim>) → GridWrap<T, dim>

Scatter 2D surfaces

class tamaas._tamaas.mpi.**sequential**

Bases: pybind11_object

__init__ (self: tamaas._tamaas.mpi.sequential) → None

tamaas._tamaas.mpi.**size**() → int

Returns the number of MPI processes

Module defining basic computations on fields.

tamaas._tamaas.compute.**deviatoric** (model_type: tamaas._tamaas.model_type, deviatoric: GridWrap<T, dim>, field: GridWrap<T, dim>) → None

Compute the deviatoric part of a tensor field

`tamaas._tamaas.compute.eigenvalues` (*model_type: tamaas._tamaas.model_type, eigenvalues_out: GridWrap<T, dim>, field: GridWrap<T, dim>*) → None

Compute eigenvalues of a tensor field

`tamaas._tamaas.compute.from_voigt` (*arg0: GridWrap<T, dim>*) → GridWrap<T, dim>

Convert a 3D tensor field to voigt notation

`tamaas._tamaas.compute.to_voigt` (*arg0: GridWrap<T, dim>*) → GridWrap<T, dim>

Convert a 3D tensor field to voigt notation

`tamaas._tamaas.compute.von_mises` (*model_type: tamaas._tamaas.model_type, von_mises: GridWrap<T, dim>, field: GridWrap<T, dim>*) → None

Compute the Von Mises invariant of a tensor field

10.1.3 Tamaas Dumpers for Model

Dumpers for the class `Model`.

class `tamaas.dumpers.JSONDumper` (*file_descriptor: Union[str, PathLike[str], IOBase]*)

Bases: `ModelDumper`

Dumper to JSON.

__init__ (*file_descriptor: Union[str, PathLike[str], IOBase]*)

Construct with file handle.

dump (*model: Model*)

Dump model.

classmethod read (*fd: IO[str]*)

Read model from file.

class `tamaas.dumpers.FieldDumper` (*basename: Union[str, PathLike[str]], *fields, **kwargs*)

Bases: `ModelDumper`

Abstract dumper for python classes using fields.

extension = ''

name_format = '{basename}{postfix}.{extension}'

__init__ (*basename: Union[str, PathLike[str]], *fields, **kwargs*)

Construct with desired fields.

add_field (*field: str*)

Add another field to the dump.

get_fields (*model: Model*)

Get the desired fields.

dump (*model: Model*)

Dump model.

classmethod read (*file_descriptor: Union[str, PathLike[str], IOBase]*)

Read model from file.

```
classmethod read_sequence (glob_pattern)
    Read models from a file sequence.

property file_path
    Get the default filename.

class tamaas.dumpers.NumpyDumper (basename: Union[str, PathLike[str]], *fields, **kwargs)
    Bases: FieldDumper
    Dumper to compressed numpy files.

    extension = 'npz'

    classmethod read (file_descriptor: Union[str, PathLike[str], IOBase])
        Create model from Numpy file.

    __init__ (basename: Union[str, PathLike[str]], *fields, **kwargs)
        Construct with desired fields.

    add_field (field: str)
        Add another field to the dump.

    dump (model: Model)
        Dump model.

    property file_path
        Get the default filename.

    get_fields (model: Model)
        Get the desired fields.

    name_format = '{basename}{postfix}.{extension}'

    classmethod read_sequence (glob_pattern)
        Read models from a file sequence.

class tamaas.dumpers.H5Dumper (basename: Union[str, PathLike[str]], *fields, **kwargs)
    Bases: FieldDumper
    Dumper to HDF5 file format.

    extension = 'h5'

    __init__ (basename: Union[str, PathLike[str]], *fields, **kwargs)
        Construct with desired fields.

    classmethod read (file_descriptor: Union[str, PathLike[str], IOBase])
        Create model from HDF5 file.

    add_field (field: str)
        Add another field to the dump.

    dump (model: Model)
        Dump model.

    property file_path
        Get the default filename.

    get_fields (model: Model)
        Get the desired fields.
```

```
name_format = '{basename}{postfix}.{extension}'

classmethod read_sequence (glob_pattern)
    Read models from a file sequence.

class tamaas.dumpers.UVWDumper (basename: Union[str, PathLike[str]], *fields, **kwargs)
    Bases: FieldDumper
    Dumper to VTK files for elasto-plastic calculations.
    extension = 'vtr'

    __init__ (basename: Union[str, PathLike[str]], *fields, **kwargs)
        Construct with desired fields.

    add_field (field: str)
        Add another field to the dump.

    dump (model: Model)
        Dump model.

    property file_path
        Get the default filename.

    get_fields (model: Model)
        Get the desired fields.

    name_format = '{basename}{postfix}.{extension}'

    classmethod read (file_descriptor: Union[str, PathLike[str], IOBase])
        Read model from file.

    classmethod read_sequence (glob_pattern)
        Read models from a file sequence.

class tamaas.dumpers.UVWGroupDumper (basename: Union[str, PathLike[str]], *fields, **kwargs)
    Bases: FieldDumper
    Dumper to ParaViewData files.
    extension = 'pvd'

    __init__ (basename: Union[str, PathLike[str]], *fields, **kwargs)
        Construct with desired fields.

    add_field (field: str)
        Add another field to the dump.

    dump (model: Model)
        Dump model.

    property file_path
        Get the default filename.

    get_fields (model: Model)
        Get the desired fields.

    name_format = '{basename}{postfix}.{extension}'
```

classmethod **read** (*file_descriptor: Union[str, PathLike[str], IOBase]*)

Read model from file.

classmethod **read_sequence** (*glob_pattern*)

Read models from a file sequence.

10.1.4 Tamaas Nonlinear solvers

Nonlinear solvers for plasticity problems.

Solvers in this module use `scipy.optimize` to solve the implicit non-linear equation for plastic deformations with fixed contact pressures.

exception `tamaas.nonlinear_solvers.NLNoConvergence`

Bases: `RuntimeError`

Convergence not reached exception.

__init__ (**args, **kwargs*)

args

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `tamaas.nonlinear_solvers.DFSANESolver` (*residual, model=None, callback=None*)

Bases: `ScipySolver`

Solve using a spectral residual jacobianless method.

See [doi:10.1090/S0025-5718-06-01840-0](https://doi.org/10.1090/S0025-5718-06-01840-0) for details on method and the relevant Scipy [documentation](#) for details on parameters.

__init__ (*residual, model=None, callback=None*)

Construct nonlinear solver with residual.

Parameters

- **residual** – plasticity residual object
- **model** – Deprecated
- **callback** – passed on to the Scipy solver

beforeSolve (*self: tamaas._tamaas.EPSolver*) → `None`

getResidual (*self: tamaas._tamaas.EPSolver*) → `tamaas._tamaas.Residual`

getStrainIncrement (*self: tamaas._tamaas.EPSolver*) → `GridBaseWrap<T>`

reset ()

Set solution vector to zero.

setToleranceManager (*self: tamaas._tamaas.EPSolver, arg0: tamaas._tamaas._tolerance_manager*) → `None`

solve ()

Solve $R(\delta\epsilon) = 0$ using a Scipy function.

property `tolerance`

updateState (*self*: [tamaas._tamaas.EPSolver](#)) → None

[tamaas.nonlinear_solvers.DFSANECXXSolver](#)

alias of [_DFSANESolver](#)

class [tamaas.nonlinear_solvers.NewtonKrylovSolver](#) (*residual*, *model=None*, *callback=None*)

Bases: [ScipySolver](#)

Solve using a finite-difference Newton-Krylov method.

__init__ (*residual*, *model=None*, *callback=None*)

Construct nonlinear solver with residual.

Parameters

- **residual** – plasticity residual object
- **model** – Deprecated
- **callback** – passed on to the Scipy solver

beforeSolve (*self*: [tamaas._tamaas.EPSolver](#)) → None

getResidual (*self*: [tamaas._tamaas.EPSolver](#)) → [tamaas._tamaas.Residual](#)

getStrainIncrement (*self*: [tamaas._tamaas.EPSolver](#)) → [GridBaseWrap<T>](#)

reset ()

Set solution vector to zero.

setToleranceManager (*self*: [tamaas._tamaas.EPSolver](#), *arg0*: [tamaas._tamaas._tolerance_manager](#)) → None

solve ()

Solve $R(\delta\epsilon) = 0$ using a Scipy function.

property tolerance

updateState (*self*: [tamaas._tamaas.EPSolver](#)) → None

[tamaas.nonlinear_solvers.ToleranceManager](#) (*start*, *end*, *rate*)

Decorate solver to manage tolerance of non-linear solve step.

10.1.5 Tamaas utilities

Convenience utilities.

[tamaas.utils.log_context](#) (*log_level*: [LogLevel](#))

Context manager to easily control Tamaas' logging level.

[tamaas.utils.publications](#) (*format_str*='{pub.citation}\n\t{pub.doi}')

Print publications associated with objects in use.

[tamaas.utils.load_path](#) (*solver*: [ContactSolver](#), *loads*: [Iterable\[Union\[float, ndarray\]\]](#), *verbose*: *bool = False*, *callback=None*) → [Iterable\[Model\]](#)

Generate model objects solutions for a sequence of applied loads.

Parameters

- **solver** – a contact solver object

- **loads** – an iterable sequence of loads
- **verbose** – print info output of solver
- **callback** – a callback executed after the yield

`tamaas.utils.seeded_surfaces` (*generator: Union[SurfaceGenerator1D, SurfaceGenerator2D], seeds: Iterable[int]*) → `Iterable[ndarray]`

Generate rough surfaces with a prescribed seed sequence.

Parameters

- **generator** – surface generator object
- **seeds** – random seed sequence

`tamaas.utils.hertz_surface` (*system_size: Iterable[float], shape: Iterable[int], radius: float*) → `ndarray`

Construct a parabolic surface.

Parameters

- **system_size** – size of the domain in each direction
- **shape** – number of points in each direction
- **radius** – radius of surface

`tamaas.utils.radial_average` (*x: ndarray, y: ndarray, values: ndarray, r: ndarray, theta: ndarray, method: str = 'linear', endpoint: bool = False*) → `ndarray`

Compute the radial average of a 2D field.

Averages radially for specified r values. See `scipy.interpolate.RegularGridInterpolator` for more details.

10.2 C++ API

```
template<typename Iterator>
```

```
struct acc_range
```

```
    #include <accumulator.hh> Range for convenience.
```

Public Functions

```
inline Iterator begin () const
```

```
inline Iterator end () const
```

Public Members

Iterator **_begin**

Iterator **_end**

```
template<model_type type, typename Source, typename = std::enable_if_t<is_proxy<Source>::value>>
```

```
class Accumulator
```

```
    #include <accumulator.hh> Accumulation integral manager.
```

Public Types

```
enum class direction
```

Direction flag.

Values:

enumerator **forward**

enumerator **backward**

Public Functions

```
inline Accumulator ()
```

Constructor.

```
inline void makeUniformMesh (UInt N, Real domain_size)
```

Initialize uniform mesh.

```
inline const std::vector<Real> &nodePositions () const
```

```
inline acc_range<iterator<direction::forward>> forward (std::vector<BufferType> &nvalues, const Grid<Real,  
                                                         trait::boundary_dimension> &wvectors)
```

Prepare forward loop.

```
inline acc_range<iterator<direction::backward>> backward (std::vector<BufferType> &nvalues, const  
                                                         Grid<Real, trait::boundary_dimension>  
                                                         &wvectors)
```

Prepare backward loop.

```
inline void reset (std::vector<BufferType> &nvalues, const Grid<Real, trait::boundary_dimension>  
                  &wvectors)
```

```
inline std::vector<BufferType> &nodeValue ()
```

```
inline const Grid<Real, trait::boundary_dimension> &waveVectors ()
```

```
inline auto elements ()
```

Create a range over the elements in the mesh.

Private Types

```
using trait = model_type_traits<type>
```

```
using BufferType = GridHermitian<Real, trait::boundary_dimension>
```

Private Members

```
std::array<BufferType, 2> accumulator
```

```
std::vector<Real> node_positions
```

```
std::vector<BufferType> *node_values
```

```
const Grid<Real, trait::boundary_dimension> *wavevectors
```

```
class AdhesionFunctional : public tamaas::functional::Functional
    #include <adhesion_functional.hh> Functional class for adhesion energy.
    Subclassed by tamaas::functional::ExponentialAdhesionFunctional, tamaas::functional::MaugisAdhesionFunctional, tamaas::functional::SquaredExponentialAdhesionFunctional
```

Public Functions

```
inline const std::map<std::string, Real> &getParameters () const
    Get parameters.
```

```
inline void setParameters (std::map<std::string, Real> other)
    Set parameters.
```

Protected Functions

```
inline AdhesionFunctional (const GridBase<Real> &surface)
    Constructor.
```

Protected Attributes

```
GridBase<Real> surface
```

```
std::map<std::string, Real> parameters
```

```
class AndersonMixing : public tamaas::EPICSolver
    #include <anderson.hh>
```

Public Functions

AndersonMixing (*ContactSolver* &csolver, *EPSolver* &epsolver, *Real* tolerance = 1e-10, *UInt* memory = 5)

virtual *Real* **solve** (const std::vector<*Real*> &load) override

Private Types

using **memory_t** = std::deque<*GridBase*<*Real*>>

Private Members

UInt **M**

Private Static Functions

static std::vector<*Real*> **computeGamma** (const *memory_t* &residual)

static *GridBase*<*Real*> **mixingUpdate** (*GridBase*<*Real*> x, std::vector<*Real*> gamma, const *memory_t* &memory, const *memory_t* &residual_memory, *Real* relaxation)

template<size_t **nargs**>

struct **Apply**

#include <apply.hh> Helper function for application of a functor on a thrust::tuple.

Public Static Functions

template<typename **Functor**, typename **Tuple**, typename ...**Args**>
static inline auto **apply** (*Functor* &&func, *Tuple* &&t, *Args*&&... args) -> decltype(*Apply*<*nargs* - 1>::apply(std::forward<*Functor*>(func), std::forward<*Tuple*>(t), thrust::get<*nargs* - 1>(t), std::forward<*Args*>(args)...))

template<>

struct **Apply**<0>

#include <apply.hh>

Public Static Functions

template<typename **Functor**, typename **Tuple**, typename ...**Args**>
static inline auto **apply** (*Functor* &&func, *Tuple*&&, *Args*&&... args) -> decltype(func(std::forward<*Args*>(args)...))

template<typename **Functor**, typename **ret_type** = void>

class **ApplyFunctor**

#include <apply.hh> Helper class for functor application in thrust.

Public Functions

```
inline ApplyFunctor (const Functor &functor)
```

```
inline ApplyFunctor (const ApplyFunctor &o)
```

```
template<typename Tuple>
```

```
inline ret_type operator () ( Tuple &&t) const
```

Private Members

```
const Functor &functor
```

```
template<typename T>
```

```
class arange
```

```
    #include <loop.hh> Helper class to count iterations within lambda-loop.
```

Public Types

```
using it_type = thrust::counting_iterator<T>
```

```
using reference = typename it_type::reference
```

Public Functions

```
inline arange (T start, T size)
```

```
inline it_type begin (UInt = 1) const
```

```
inline it_type end (UInt = 1) const
```

```
inline UInt getNbComponents () const
```

Private Members

```
T start
```

```
T range_size
```

```
template<typename T>
```

```
struct Array
```

```
    #include <array.hh> Generic storage class with wrapping capacities.
```

Public Functions

Array () = default

Default.

inline **Array** (*UInt* size)

Empty array of given size.

inline **Array** (const *Array* &v)

Copy constructor (deep)

inline **Array** (*Array* &&v) noexcept

Move constructor (transfers data ownership)

inline **Array** (*T* *data, *UInt* size) noexcept

Wrap array on data.

inline **Array** (*span*<*T*> view) noexcept

Wrap on span.

inline **~Array** ()

Destructor.

inline *Array* &**operator=** (const *Array* &v)

Copy operator.

inline *Array* &**operator=** (*Array* &&v) noexcept

Move operator.

inline *Array* &**operator=** (*span*<*T*> v) noexcept

Wrap on view.

inline void **wrap** (const *Array* &other) noexcept

Wrap array.

inline void **wrap** (*span*<*T*> view) noexcept

Wrap view.

inline void **wrap** (*T* *data, *UInt* size) noexcept

Wrap a memory pointer.

inline const *T* ***data** () const

Data pointer access (const)

inline *T* ***data** ()

Data pointer access (non-const)

inline void **resize** (*UInt* new_size, const *T* &value = *T*())

Resize array.

inline void **reserve** (*UInt* size)

Reserve storage space.

inline *T* &**operator** [] (*UInt* i)

Access operator.

inline const *T* &**operator** [] (*UInt* i) const

Access operator (const)

```
inline UInt size () const  
    Get size of array.  
inline span<T> view () const
```

Private Members

```
span<T> view_  
  
span<T>::size_type reserved_ = 0  
  
bool wrapped_ = false  
  
Allocator<T> alloc_
```

```
class assertion_error : public invalid_argument  
    #include <errors.hh>
```

```
class BeckTeboulle : public tamaas::Kato  
    #include <beck_teboulle.hh>
```

Public Functions

```
BeckTeboulle (Model &model, const GridBase<Real> &surface, Real tolerance, Real mu)  
    Constructor.  
  
Real solve (GridBase<Real> &g0)  
    Solve.
```

Private Functions

```
template<model_type type>  
Real solveTpl (GridBase<Real> &g0)  
    Template for solve function.
```

```
class BEEngine  
    #include <be_engine.hh> Boundary equation engine class. Solves the Neumann/Dirichlet problem This class should  
    be dimension and model-type agnostic.  
  
    Subclassed by tamaas::BEEngineTpl<type >
```

Public Functions

```
inline BEEngine (Model *model)

virtual ~BEEngine () = default
    Destructor.

virtual void solveNeumann (GridBase<Real> &neumann, GridBase<Real> &dirichlet) const = 0
    Solve Neumann problem (expects boundary data)

virtual void solveDirichlet (GridBase<Real> &dirichlet, GridBase<Real> &neumann) const = 0
    Solve Dirichlet problem (expects boundary data)

virtual void registerNeumann () = 0
    Register neumann operator.

virtual void registerDirichlet () = 0
    Register dirichlet operator.

inline const Model &getModel () const
    Get model.

inline Real getNeumannNorm ()
    Compute L2 norm of influence functions.
```

Protected Attributes

Model ***model**

std::map<*IntegralOperator::kind*, std::shared_ptr<*IntegralOperator*>> **operators**

template<*model_type* **type**>

```
class BEEngineTmpl : public tamaas::BEEngine
    #include <be_engine.hh>
```

Public Functions

```
inline BEEngineTmpl (Model *model)

virtual void solveNeumann (GridBase<Real> &neumann, GridBase<Real> &dirichlet) const override
    Solve Neumann problem (expects boundary data)

virtual void solveDirichlet (GridBase<Real> &dirichlet, GridBase<Real> &neumann) const override
    Solve Dirichlet problem (expects boundary data)

virtual void registerNeumann () override
    Register neumann operator.

virtual void registerDirichlet () override
    Register dirichlet operator.

template<UInt m, UInt j>

struct boundary_fft_helper
```

Public Static Functions

```
template<typename Buffer, typename Out>
static inline void backwardTransform (FFTEngine &e, Buffer &&buffer, Out &&out)

template<UInt m>

struct boundary_fft_helper<m, m>
```

Public Static Functions

```
template<typename Buffer, typename Out>
static inline void backwardTransform (FFTEngine &e, Buffer &&buffer, Out &&out)

template<model_type type, UInt derivative>

class Boussinesq : public tamaas::VolumePotential<type>
    #include <boussinesq.hh> Boussinesq tensor.
```

Public Functions

```
Boussinesq (Model *model)
    Constructor.

virtual void apply (GridBase<Real> &source, GridBase<Real> &out) const override
    Apply the Boussinesq operator.
```

Protected Functions

```
void initialize (UInt source_components, UInt out_components)
```

Private Types

```
using trait = model_type_traits<type>

using parent = VolumePotential<type>

template<UInt dim, UInt derivative_order>

class Boussinesq
    #include <influence.hh> Class for the Boussinesq tensor.

template<>

class Boussinesq<3, 0>
    #include <influence.hh> Subclassed by tamaas::influence::Boussinesq<3, 1>
```

Public Functions

```
inline Boussinesq (Real mu, Real nu)
```

Constructor.

```
template<bool apply_q_power = false, typename ST>
```

```
inline Vector<Complex, dim> applyU0 (const StaticVector<Complex, ST, dim> &t, const VectorProxy<const  
Real, dim - 1> &q) const
```

```
template<bool apply_q_power = false, typename ST>
```

```
inline Vector<Complex, dim> applyU1 (const StaticVector<Complex, ST, dim> &t, const VectorProxy<const  
Real, dim - 1> &q) const
```

Protected Attributes

```
const Real mu
```

```
const Real nu
```

```
const Real lambda
```

Protected Static Attributes

```
static constexpr UInt dim = 3
```

```
static constexpr UInt order = 0
```

```
template<>
```

```
class Boussinesq<3, 1> : protected tamaas::influence::Boussinesq<3, 0>
```

```
    #include <influence.hh> Boussinesq first gradient.
```

Public Functions

```
template<bool apply_q_power = false, typename ST>
```

```
inline Matrix<Complex, dim, dim> applyU0 (const StaticVector<Complex, ST, dim> &t, const  
VectorProxy<const Real, dim - 1> &q) const
```

```
template<bool apply_q_power = false, typename ST>
```

```
inline Matrix<Complex, dim, dim> applyU1 (const StaticVector<Complex, ST, dim> &t, const  
VectorProxy<const Real, dim - 1> &q) const
```

Protected Types

```
using parent = Boussinesq<3, 0>
```

Protected Static Attributes

```
static constexpr UInt dim = parent::dim
```

```
static constexpr UInt order = parent::order + 1
```

```
template<model_type type, typename boussinesq_t>
```

```
struct BoussinesqHelper
```

```
    #include <boussinesq_helper.hh>
```

Public Types

```
using trait = model_type_traits<type>
```

```
using BufferType = GridHermitian<Real, bdim>
```

```
using source_t = typename KelvinTrait<boussinesq_t>::source_t
```

```
using out_t = typename KelvinTrait<boussinesq_t>::out_t
```

Public Functions

```
template<bool apply_q_power>
```

```
inline void apply (BufferType &tractions, std::vector<BufferType> &out, const Grid<Real, bdim>  
                  &wavevectors, Real domain_size, const boussinesq_t &boussinesq)
```

```
template<bool apply_q_power>
```

```
inline void apply (BufferType &tractions, BufferType &out, UInt layer, const Grid<Real, bdim> &wavevectors,  
                  UInt discretization, Real domain_size, const boussinesq_t &boussinesq)
```

```
inline void makeFundamentalModeGreatAgain (BufferType&, std::vector<BufferType>&,  
                                           influence::ElasticHelper<dim>&)
```

```
template<typename ST>
```

```
inline void makeFundamentalModeGreatAgain (StaticVector<Complex, ST, dim>&, out_t&,  
                                           influence::ElasticHelper<dim>&)
```

Public Static Attributes

static constexpr *UInt* **dim** = *trait*::dimension

static constexpr *UInt* **bdim** = *trait*::boundary_dimension

Protected Attributes

Accumulator<*type*, *source_t*> **accumulator**

really only here for mesh

template<*UInt* **dim**>

class **Cluster**

#include <flood_fill.hh>

Public Functions

Cluster (*Point* start, const *Grid*<bool, *dim*> &map, *Grid*<bool, *dim*> &visited, bool diagonal)

Constructor.

Cluster (const *Cluster* &other)

Copy constructor.

Cluster () = default

Default constructor.

inline *UInt* **getArea** () const

Get area of cluster.

inline *UInt* **getPerimeter** () const

Get perimeter of cluster.

inline const auto &**getPoints** () const

Get contact points.

BBox **boundingBox** () const

Get bounding box.

std::array<*Int*, *dim*> **extent** () const

Get bounding box extent.

auto **getNextNeighbors** (const std::array<*Int*, *dim*> &p)

Assign next neighbors.

auto **getDiagonalNeighbors** (const std::array<*Int*, *dim*> &p)

Assign diagonal neighbors.

auto **getNextNeighbors** (const std::array<*Int*, 1> &p)

auto **getDiagonalNeighbors** (const std::array<*Int*, 1>&)

```
auto getNextNeighbors (const std::array<Int, 2> &p)
auto getDiagonalNeighbors (const std::array<Int, 2> &p)
auto getNextNeighbors (const std::array<Int, 3> &p)
auto getDiagonalNeighbors (const std::array<Int, 3> &p)
```

Private Types

```
using Point = std::array<Int, dim>

using BBox = std::pair<std::array<Int, dim>, std::array<Int, dim>>
```

Private Members

```
std::vector<Point> points
    List of points in the cluster.

UInt perimeter = 0
    Perimeter size (number of segments)
```

```
struct comm
    #include <mpi_interface.hh>
```

Public Static Attributes

```
static comm world

template<class Compute_t>
class ComputeOperator : public tamaas::IntegralOperator
```

Public Functions

```
inline ComputeOperator (Model *model)
    Constructor.

inline virtual IntegralOperator::kind getKind () const override
    Kind.

inline virtual model_type getType () const override
    Type.

inline virtual void updateFromModel () override
    Update any data dependent on model parameters.
```

```
inline virtual void apply (GridBase<Real> &in, GridBase<Real> &out) const override  
    Apply functor.
```

```
class Condat : public tamaas::Kato  
    #include <condat.hh>
```

Public Functions

```
Condat (Model &model, const GridBase<Real> &surface, Real tolerance, Real mu)  
    Constructor.
```

```
Real solve (GridBase<Real> &p0, Real grad_step)  
    Solve.
```

```
template<model_type type>  
Real solveTpl (GridBase<Real> &p0, Real grad_step)  
    Template for solve function.
```

```
template<UInt comp>  
void updateGap (Real sigma, Real grad_step, GridBase<Real> &q)  
    Update gap.
```

```
template<UInt comp>  
void updateLagrange (GridBase<Real> &q, GridBase<Real> &p0)  
    Update Lagrange multiplier q.
```

Private Members

```
std::unique_ptr<GridBase<Real>>> pressure_old = nullptr
```

```
class ContactSolver  
    #include <contact_solver.hh> Subclassed by tamaas::Kato, tamaas::PolonskyKeerRey
```

Public Functions

```
ContactSolver (Model &model, const GridBase<Real> &surface, Real tolerance)  
    Constructor.
```

```
virtual ~ContactSolver () = default  
    Destructor.
```

```
virtual void printState (UInt iter, Real cost_f, Real error) const  
    Print state of solve.
```

```
virtual void logIteration (UInt iter, Real cost_f, Real error) const  
    Log iteration info.
```

```
inline auto getMaxIterations () const  
    Get maximum number of iterations.
```

```

inline void setMaxIterations (UInt n)
    Set maximum number of iterations.

inline auto getDumpFrequency () const
    Get dump_frequency.

inline void setDumpFrequency (UInt n)
    Set dump_frequency.

void addFunctionalTerm (std::shared_ptr<functional::Functional> func)
    Add term to functional.

inline const functional::Functional &getFunctional () const
    Returns functional object.

void setFunctional (std::shared_ptr<functional::Functional> func)
    Sets functional sum to a single term.

inline virtual Real solve (std::vector<Real>)
    Solve for a mean traction vector.

inline virtual Real solve (Real load)
    Solve for normal pressure.

TAMAAS_ACCESSOR (tolerance, Real, Tolerance)
    Accessor for tolerance.

inline GridBase<Real> &getSurface ()

inline Model &getModel ()

```

Protected Attributes

Model &**model**

GridBase<*Real*> **surface**

std::shared_ptr<*GridBase*<*Real*>> **_gap** = nullptr

functional::MetaFunctional **functional**

Real **tolerance**

UInt **max_iterations** = 1000

Real **surface_stddev**

UInt **dump_frequency** = 100

```

class CuFFTEngine : public tamaas::FFTEngine
    #include <cufft_engine.hh>

```

Public Functions

inline explicit **CuFFTEngine** (unsigned int flags = FFTW_ESTIMATE) noexcept
Initialize with flags.

inline virtual void **forward** (const *Grid<Real, 1>* &real, *GridHermitian<Real, 1>* &spectral) override
Execute a forward plan on real and spectral 1D.

inline virtual void **forward** (const *Grid<Real, 2>* &real, *GridHermitian<Real, 2>* &spectral) override
Execute a forward plan on real and spectral 2D.

inline virtual void **backward** (*Grid<Real, 1>* &real, *GridHermitian<Real, 1>* &spectral) override
Execute a backward plan on real and spectral 1D.

inline virtual void **backward** (*Grid<Real, 2>* &real, *GridHermitian<Real, 2>* &spectral) override
Execute a backward plan on real and spectral 2D.

inline unsigned int **flags** () const

Public Static Functions

static inline auto **cast** (*Complex* *data)
Cast to FFTW complex type.

static inline auto **cast** (const *Complex* *data)

Protected Attributes

unsigned int **_flags**
FFTW flags.

std::map<key_t, *plan_t*> **plans**
plans corresponding to signatures

Private Types

using **plan_t** = std::pair<*cufft::plan*, *cufft::plan*>

Private Functions

template<*UInt dim*>
void **forwardImpl** (const *Grid<Real, dim>* &real, *GridHermitian<Real, dim>* &spectral)
Perform forward (R2C) transform.

template<*UInt dim*>
void **backwardImpl** (*Grid<Real, dim>* &real, const *GridHermitian<Real, dim>* &spectral)
Perform backward (C2R) transform.

plan_t &**getPlans** (key_t key)

Return the plans pair for a given transform signature.

template<bool **upper**>

struct **cutoff_functor**

#include <kelvin_helper.hh>

Public Functions

inline void **operator** () (*VectorProxy*<const *Real*, bdim> qv, source_t wj0, source_t wj1, out_t out_i) const

Public Members

Real **r**

Real **xc**

Real **dx**

Real **cutoff**

kelvin_t **kelvin**

class **DCFFT** : public *tamaas::Westergaard*<*model_type::basic_2d*, *IntegralOperator::neumann*>

#include <dcfft.hh> Non-periodic *Boussinesq* operator, computed with padded FFT.

Public Functions

DCFFT (*Model* *model)

virtual void **apply** (*GridBase*<*Real*> &input, *GridBase*<*Real*> &output) const override

Apply influence coefficients in Fourier domain.

Private Types

using **trait** = *model_type_traits*<*model_type::basic_2d*>

Private Functions

void **initInfluence** ()

Init influence with real-space square patch solution.

Private Members

mutable *Grid*<*Real*, *bdim*> **extended_buffer**

Private Static Attributes

static constexpr *UInt* **dim** = *trait*::dimension

static constexpr *UInt* **bdim** = *trait*::boundary_dimension

static constexpr *UInt* **comp** = *trait*::components

template<*UInt* **derivative**>

struct **derivative_traits**

#include <volume_potential.hh> Trait type for component management.

template<>

struct **derivative_traits**<0>

#include <volume_potential.hh>

Public Static Attributes

template<*model_type* **type**>

static constexpr *UInt* **source_components** = *model_type_traits*<*type*>::components

template<*model_type* **type**>

static constexpr *UInt* **out_components** = *model_type_traits*<*type*>::components

template<>

struct **derivative_traits**<1>

#include <volume_potential.hh>

Public Static Attributes

```

template<model_type type>

static constexpr UInt source_components = model_type_traits<type>::voigt

template<model_type type>

static constexpr UInt out_components = model_type_traits<type>::components

template<>

struct derivative_traits<2>
    #include <volume_potential.hh>

```

Public Static Attributes

```

template<model_type type>

static constexpr UInt source_components = model_type_traits<type>::voigt

template<model_type type>

static constexpr UInt out_components = model_type_traits<type>::voigt

struct Deviatoric
    #include <computes.hh> Compute deviatoric of tensor field.

```

Public Static Functions

```

template<UInt dim>

static inline void call (Grid<Real, dim> &dev, const Grid<Real, dim> &field)

```

```

class DFSANESolver : public tamaas::EPSolver

```

```

    #include <dfsane_solver.hh> Derivative-free non-linear solver.

```

This algorithm is based on W. La Cruz, J. Martínez, and M. Raydan, “Spectral residual method without gradient information for solving large-scale nonlinear systems of equations,” Math. Comp., vol. 75, no. 255, pp. 1429–1448, 2006, doi: 10.1090/S0025-5718-06-01840-0.

The same algorithm is available in `scipy.optimize`, but this version is robustly parallel by default (i.e. does not depend on BLAS’s parallelism and is future-proof for MPI parallelism).

Public Functions

```

DFSANESolver (Residual &residual)

```

```

virtual void solve () override

```

```

TAMAAS_ACCESSOR (max_iterations, UInt, MaxIterations)

```

Protected Functions

Real **computeSpectralCoeff** (const std::pair<*Real*, *Real*> &bounds)

void **computeSearchDirection** (*Real* sigma)

void **lineSearch** (*Real* eta_k)

Protected Attributes

GridBase<*Real*> **search_direction**

GridBase<*Real*> **previous_residual**

GridBase<*Real*> **current_x**

GridBase<*Real*> **delta_x**

GridBase<*Real*> **delta_residual**

std::deque<*Real*> **previous_merits**

std::function<*Real*(*UInt*)> **eta**

UInt **max_iterations** = 100

Protected Static Functions

static *Real* **computeAlpha** (*Real* alpha, *Real* f, *Real* fk, const std::pair<*Real*, *Real*> &bounds)

struct **Eigenvalues**

#include <computes.hh> Compute eigenvalues of a symmetric matrix field.

Public Static Functions

template<*UInt* **dim**>

static inline void **call** (*Grid*<*Real*, *dim*> &eigs, const *Grid*<*Real*, *dim*> &field)

class **ElasticFunctional** : public *tamaas::functional::Functional*

#include <elastic_functional.hh> Generic functional for elastic energy.

Subclassed by *tamaas::functional::ElasticFunctionalGap*, *tamaas::functional::ElasticFunctionalPressure*

Public Functions

inline **ElasticFunctional** (const *IntegralOperator* &op, const *GridBase<Real>* &surface)

Protected Attributes

const *IntegralOperator* &op

GridBase<Real> surface

mutable std::unique_ptr<*GridBase<Real>*>> buffer

class **ElasticFunctionalGap** : public *tamaas::functional::ElasticFunctional*
#include <elastic_functional.hh> Functional with gap as primal field.

Public Functions

virtual *Real* **computeF** (*GridBase<Real>* &gap, *GridBase<Real>* &dual) const override
 Compute functional with input gap.

virtual void **computeGradF** (*GridBase<Real>* &gap, *GridBase<Real>* &gradient) const override
 Compute functional gradient with input gap.

inline **ElasticFunctional** (const *IntegralOperator* &op, const *GridBase<Real>* &surface)

class **ElasticFunctionalPressure** : public *tamaas::functional::ElasticFunctional*
#include <elastic_functional.hh> Functional with pressure as primal field.

Public Functions

virtual *Real* **computeF** (*GridBase<Real>* &pressure, *GridBase<Real>* &dual) const override
 Compute functional with input pressure.

virtual void **computeGradF** (*GridBase<Real>* &pressure, *GridBase<Real>* &gradient) const override
 Compute functional gradient with input pressure.

inline **ElasticFunctional** (const *IntegralOperator* &op, const *GridBase<Real>* &surface)

template<U*Int* dim>

struct **ElasticHelper**

#include <influence.hh> Functor to apply *Hooke's* tensor.

Public Functions

inline **ElasticHelper** (*Real* mu, *Real* nu)

template<typename **DT**, typename **ST**>

inline *Matrix*<std::remove_cv_t<*DT*>, *dim*, *dim*> **operator** () (const *StaticMatrix*<*DT*, *ST*, *dim*, *dim*> &gradu) const

template<typename **DT**, typename **ST**>

inline *SymMatrix*<std::remove_cv_t<*DT*>, *dim*> **operator** () (const *StaticSymMatrix*<*DT*, *ST*, *dim*> &eps) const

template<typename **DT**, typename **ST**>

inline *Matrix*<std::remove_cv_t<*DT*>, *dim*, *dim*> **inverse** (const *StaticMatrix*<*DT*, *ST*, *dim*, *dim*> &sigma) const

Public Members

const *Real* **mu**

const *Real* **nu**

const *Real* **lambda**

class **EPICSolver**

#include <epic.hh> Subclassed by *tamaas::AndersonMixing*

Public Functions

EPICSolver (*ContactSolver* &csolver, *EPSolver* &epsolver, *Real* tolerance = 1e-10, *Real* relaxation = 0.3)

Constructor.

virtual *Real* **solve** (const std::vector<*Real*> &load)

Real **acceleratedSolve** (const std::vector<*Real*> &load)

Real **computeError** (const *GridBase*<*Real*> ¤t, const *GridBase*<*Real*> &prev, *Real* factor) const

void **fixedPoint** (*GridBase*<*Real*> &result, const *GridBase*<*Real*> &x, const *GridBase*<*Real*> &initial_surface, std::vector<*Real*> load)

template<*model_type* **type**>

void **setViews** ()

TAMAAS_ACCESSOR (*tolerance*, *Real*, Tolerance)

TAMAAS_ACCESSOR (*relaxation*, *Real*, Relaxation)

TAMAAS_ACCESSOR (*max_iterations*, *UInt*, MaxIterations)

inline const *Model* &**getModel** () const

Protected Attributes

GridBase<Real> **surface**

corrected surface

GridBase<Real> **pressure**

current pressure

std::unique_ptr<*GridBase<Real>*> **residual_disp**

plastic residual disp

std::unique_ptr<*GridBase<Real>*> **pressure_inc**

pressure increment

ContactSolver &**csolver**

EPSolver &**epsolver**

Real **tolerance**

Real **relaxation**

UInt **max_iterations** = 1000

class **EPSolver**

#include <ep_solver.hh> Subclassed by *tamaas::DFSANESolver*

Public Functions

EPSolver (*Residual* &residual)

Constructor.

virtual ~**EPSolver** () = default

Destructor.

virtual void **solve** () = 0

virtual void **updateState** ()

virtual void **beforeSolve** ()

inline *GridBase<Real>* &**getStrainIncrement** ()

inline *Residual* &**getResidual** ()

inline *Real* **getTolerance** () const

inline void **setTolerance** (*Real* tol)

inline void **setToleranceManager** (*ToleranceManager* manager)

Protected Attributes

`std::shared_ptr<GridBase<Real>> _x`

Residual & `_residual`

ToleranceManager `abs_tol = { 1e-9, 1e-9, 1 }`

class **Exception** : public exception
#include <tamaas.hh> Generic exception class.

Public Functions

inline **Exception** (std::string msg)

Constructor.

inline const char ***what** () const noexcept override

~Exception () override = default

Private Members

`std::string msg`

message of exception

class **ExponentialAdhesionFunctional** : public *tamaas::functional::AdhesionFunctional*
#include <adhesion_functional.hh> Exponential adhesion functional.

Public Functions

inline **ExponentialAdhesionFunctional** (const *GridBase<Real>* &surface)

Explicit declaration of constructor for swig.

virtual *Real* **computeF** (*GridBase<Real>* &gap, *GridBase<Real>* &pressure) const override

Compute the total adhesion energy.

virtual void **computeGradF** (*GridBase<Real>* &gap, *GridBase<Real>* &gradient) const override

Compute the gradient of the adhesion functional.

template<*UInt* interpolation_order>

struct **ExponentialElement**

#include <element.hh>

template<>

struct **ExponentialElement**<1>

#include <element.hh>

Public Static Functions

```
template<UInt shape>
static inline constexpr expolit::Polynomial<Real, 1> shapes ()
```

```
static inline constexpr auto sign (bool upper)
```

```
template<bool upper, UInt shape>
static inline constexpr auto g0 (Real q)
```

```
template<bool upper, UInt shape>
static inline constexpr auto g1 (Real q)
```

class **FFTEngine**

#include <fft_engine.hh> Subclassed by *tamaas::CuFFTEngine*, *tamaas::FFTWEngine*

Public Functions

```
virtual ~FFTEngine () noexcept = default
```

```
virtual void forward (const Grid<Real, 1> &real, GridHermitian<Real, 1> &spectral) = 0
    Execute a forward plan on real and spectral 1D.
```

```
virtual void forward (const Grid<Real, 2> &real, GridHermitian<Real, 2> &spectral) = 0
    Execute a forward plan on real and spectral 2D.
```

```
virtual void backward (Grid<Real, 1> &real, GridHermitian<Real, 1> &spectral) = 0
    Execute a backward plan on real and spectral 1D.
```

```
virtual void backward (Grid<Real, 2> &real, GridHermitian<Real, 2> &spectral) = 0
    Execute a backward plan on real and spectral 2D.
```

Public Static Functions

```
template<typename T, UInt dim, bool hermitian>
static Grid<T, dim> computeFrequencies (const std::array<UInt, dim> &sizes)
    Fill a grid with wavevector values in appropriate ordering.
```

```
template<UInt dim>
static std::vector<std::array<UInt, dim>> realCoefficients (const std::array<UInt, dim> &sizes)
    Return the grid indices that contain real coefficients (see R2C transform)
```

```
static std::unique_ptr<FFTEngine> makeEngine (unsigned int flags = FFTW_ESTIMATE)
    Instantiate an appropriate FFTEngine subclass.
```

```
template<>
static std::vector<std::array<UInt, 2>> realCoefficients (const std::array<UInt, 2> &local_sizes)
```

```
template<>
static std::vector<std::array<UInt, 1>> realCoefficients (const std::array<UInt, 1> &local_sizes)
```

Protected Types

```
using key_t = std::basic_string<UInt>
```

Protected Static Functions

```
template<UInt dim>
static key_t make_key (const Grid<Real, dim> &real, const GridHermitian<Real, dim> &spectral)
    Make a transform signature from a pair of grids.
```

```
template<typename T>
```

```
struct FFTWAllocator
```

```
    #include <fftw_allocator.hh> Class allocating SIMD aligned memory
```

Public Static Functions

```
static inline span<T> allocate (typename span<T>::size_type n) noexcept
    Allocate memory.

static inline void deallocate (span<T> view) noexcept
    Free memory.
```

```
class FFTWEngine : public tamaas::FFTEngine
```

```
    #include <fftw_engine.hh> Subclassed by tamaas::FFTWMPIEngine
```

Public Functions

```
inline explicit FFTWEngine (unsigned int flags = FFTW_ESTIMATE) noexcept
    Initialize with flags.

inline virtual void forward (const Grid<Real, 1> &real, GridHermitian<Real, 1> &spectral) override
    Execute a forward plan on real and spectral 1D.

inline virtual void forward (const Grid<Real, 2> &real, GridHermitian<Real, 2> &spectral) override
    Execute a forward plan on real and spectral 2D.

inline virtual void backward (Grid<Real, 1> &real, GridHermitian<Real, 1> &spectral) override
    Execute a backward plan on real and spectral 1D.

inline virtual void backward (Grid<Real, 2> &real, GridHermitian<Real, 2> &spectral) override
    Execute a backward plan on real and spectral 2D.

inline unsigned int flags () const
```

Public Static Functions

static inline auto **cast** (*Complex* *data)

Cast to FFTW complex type.

static inline auto **cast** (const *Complex* *data)

Protected Types

using **plan_t** = std::pair<*fftw::plan<Real>*, *fftw::plan<Real>*>>

using **complex_t** = *fftw::helper<Real>::complex*

Protected Functions

template<*UInt* **dim**>

void **forwardImpl** (const *Grid<Real, dim>* &real, *GridHermitian<Real, dim>* &spectral)

Perform forward (R2C) transform.

template<*UInt* **dim**>

void **backwardImpl** (*Grid<Real, dim>* &real, const *GridHermitian<Real, dim>* &spectral)

Perform backward (C2R) transform.

plan_t &**getPlans** (key_t key)

Return the plans pair for a given transform signature.

Protected Attributes

unsigned int **_flags**

FFTW flags.

std::map<key_t, *plan_t*> **plans**

plans corresponding to signatures

class **FFTWMPIDEngine** : public *tamaas::FFTWEngine*

#include <fftw_mpi_engine.hh>

Public Functions

inline virtual void **forward** (const *Grid<Real, 1>* &, *GridHermitian<Real, 1>* &) override

Execute a forward plan on real and spectral 1D.

inline virtual void **backward** (*Grid<Real, 1>* &, *GridHermitian<Real, 1>* &) override

Execute a backward plan on real and spectral 1D.

virtual void **forward** (const *Grid<Real, 2>* &real, *GridHermitian<Real, 2>* &spectral) override

FFTW/MPI forward (r2c) transform.

virtual void **backward** (*Grid*<*Real*, 2> &real, *GridHermitian*<*Real*, 2> &spectral) override
 FFTW/MPI backward (c2r) transform.

inline explicit **FFTWEngine** (unsigned int flags = FFTW_ESTIMATE) noexcept
 Initialize with flags.

Protected Functions

plan_t &**getPlans** (key_t key)
 Return the plans pair for a given transform signature.

Protected Attributes

std::map<key_t, *Grid*<*Real*, 2>> **workspaces**
 Buffer for real data because of FFTW/MPI layout.

Protected Static Functions

static key_t **make_key** (const *Grid*<*Real*, 2> &real, const *GridHermitian*<*Real*, 2> &spectral)
 Make a transform signature from a pair of grids.
static auto **local_size** (const key_t &key)
 Get FFTW local sizes from an hermitian grid.

struct **FieldContainer**
 #include <field_container.hh> Subclassed by *tamaas::IntegralOperator*, *tamaas::Model*

Public Types

using **Key** = std::string
template<class **T**>
using **GridBasePtr** = std::shared_ptr<*GridBase*<*T*>>

using **Value** = boost::variant<*GridBasePtr*<*Real*>, *GridBasePtr*<*UInt*>, *GridBasePtr*<*Int*>, *GridBasePtr*<*Complex*>, *GridBasePtr*<bool>>

using **FieldsMap** = std::unordered_map<*Key*, *Value*>

Public Functions

virtual **~FieldContainer** () = default

Destructor.

inline const *Value* &**at** (const *Key* &name) const

Access field pointer in const context.

inline *Value* &**operator**[] (const *Key* &name)

Access/insert new pointer.

template<class **T**>

inline auto &**field** (const *Key* &name)

Access field of given type (non-const)

template<class **T**>

inline const auto &**field** (const *Key* &name) const

Access field of given type (const)

inline decltype(auto) **fields** () const

Return field keys.

inline const auto &**fields_map** () const

Return pointer map to fields.

template<*model_type* **type**, bool **boundary**, typename **T**, template<typename, *UInt*> class **GridType** = *Grid*, class **Container** = void>

inline std::shared_ptr<*GridType*<**T**, *detail::dim_choice*<*type*, *boundary*>::value>> **request** (const *Key* &name, *Container* &&n, *UInt* nc)

Return a field with given dimension, create if not in container.

template<bool **boundary**, typename **T**, typename **Container**>

inline decltype(auto) **request** (const *Key* &name, *model_type* type, *Container* &&n, *UInt* nc)

Return a field with given dimension, create if not in container.

Private Members

FieldsMap **fields_**

template<*UInt* **dim**>

class **Filter**

#include <filter.hh> Subclassed by *tamaas::Isopowerlaw*< *dim* >, *tamaas::RegularizedPowerlaw*< *dim* >

Public Functions

Filter () = default

Default constructor.

virtual **~Filter** () = default

Destructor.

virtual void **computeFilter** (*GridHermitian*<*Real*, *dim*> &filter_coefficients) const = 0

Compute *Filter* coefficients.

Protected Functions

template<typename **T**>

void **computeFilter** (*T* &&f, *GridHermitian*<*Real*, *dim*> &filter) const

Compute filter coefficients using lambda.

class **FloodFill**

#include <flood_fill.hh>

Public Static Functions

static *List*<1> **getSegments** (const *Grid*<bool, 1> &map)

Return a list of connected segments.

static *List*<2> **getClusters** (const *Grid*<bool, 2> &map, bool diagonal)

Return a list of connected areas.

static *List*<3> **getVolumes** (const *Grid*<bool, 3> &map, bool diagonal)

Return a list of connected volumes.

Private Types

template<*UInt dim*>

using **List** = std::vector<*Cluster*<*dim*>>

template<template<typename> class **Trait**, typename ...**T**>

struct **fold_trait** : public *tamaas::detail::fold_trait_tail_rec*<true, *Trait*, *T*...>

#include <tamaas.hh>

template<bool **acc**, template<typename> class **Trait**, typename **Head**, typename ...**Tail**>

struct **fold_trait_tail_rec** : public std::integral_constant<bool, *fold_trait_tail_rec*<*acc* and *Trait*<*Head*>::value, *Trait*, *Tail*...>::value>

#include <tamaas.hh>

template<bool **acc**, template<typename> class **Trait**, typename **Head**>

struct **fold_trait_tail_rec**<*acc*, *Trait*, *Head*> : public std::integral_constant<bool, *acc* and *Trait*<*Head*>::value>

#include <tamaas.hh>

class **Functional**

#include <functional.hh> Generic functional class for the cost function of the optimization problem.

Subclassed by *tamaas::functional::AdhesionFunctional*, *tamaas::functional::ElasticFunctional*, *tamaas::functional::MetaFunctional*

Public Functions

virtual **~Functional** () = default

Destructor.

virtual *Real* **computeF** (*GridBase<Real>* &variable, *GridBase<Real>* &dual) const = 0

Compute functional.

virtual void **computeGradF** (*GridBase<Real>* &variable, *GridBase<Real>* &gradient) const = 0

Compute functional gradient.

template<*UInt* **N**, *UInt*... **ns**>

struct **get** : public *detail::get_rec<N, ns...>*

#include <static_types.hh>

template<*UInt* **n**, *UInt*... **ns**>

struct **get_rec**<0, *n*, *ns*...> : public std::integral_constant<*UInt*, *n*>

#include <static_types.hh>

template<typename **T**, *UInt* **dim**>

class **Grid** : public *tamaas::GridBase<T>*

#include <grid.hh> Multi-dimensional & multi-component array class.

This class is a container for multi-component data stored on a multi- dimensional grid.

The access function is the parenthesis operator. For a grid of dimension d, the operator takes d+1 arguments: the first d arguments are the position on the grid and the last one is the component of the stored data.

It is also possible to use the access operator with only one argument, it is then considering the grid as a flat array, accessing the given cell of the array.

Public Types

using **value_type** = *T*

using **reference** = *value_type*&

Public Functions

Grid()

Constructor by default (empty array)

template<typename **RandomAccessIterator**>

Grid (*RandomAccessIterator* begin, *RandomAccessIterator* end, *UInt* nb_components)

Constructor with shape from iterators.

template<typename **RandomAccessIterator**>

Grid (*RandomAccessIterator* begin, *RandomAccessIterator* end, *UInt* nb_components, *span*<*value_type*> data)

Construct with shape from iterators on data view.

template<typename **Container**>

inline **Grid** (*Container* &&n, *UInt* nb_components)

Constructor with container as shape.

template<typename **Container**>

inline **Grid** (*Container* &&n, *UInt* nb_components, *span*<*value_type*> data)

Constructor with shape and wrapped data.

inline **Grid** (const std::initializer_list<*UInt*> &n, *UInt* nb_components)

Constructor with initializer list.

inline **Grid** (const *Grid* &o)

Copy constructor.

inline **Grid** (*Grid* &&o) noexcept

Move constructor (transfers data ownership)

~Grid() override = default

Destructor.

void **resize** (const std::array<*UInt*, *dim*> &n)

Resize array.

void **resize** (const std::vector<*UInt*> &n)

Resize array (from std::vector)

void **resize** (std::initializer_list<*UInt*> n)

Resize array (from initializer list)

template<typename **ForwardIt**>

void **resize** (*ForwardIt* begin, *ForwardIt* end)

Resize array (from iterators)

inline *UInt* **computeSize**() const

Compute size.

inline virtual *UInt* **getDimension**() const override

Get grid dimension.

void **computeStrides**()

Compute strides.

virtual void **printself** (std::ostream &str) const

Print.

```

inline const std::array<UInt, dim> &sizes () const
    Get sizes.

inline const std::array<UInt, dim + 1> &getStrides () const
    Get strides.

template<typename ...T1>
inline std::enable_if_t<is_valid_index<T1...>::value, T&> operator () (T1... args)
    Variadic access operator (non-const)

template<typename ...T1>
inline std::enable_if_t<is_valid_index<T1...>::value, const T&> operator () (T1... args) const
    Variadic access operator.

template<std::size_t tdim>
inline T &operator () (std::array<UInt, tdim> tuple)
    Tuple index access operator.

template<std::size_t tdim>
inline const T &operator () (std::array<UInt, tdim> tuple) const

Grid &operator= (const Grid &other)

Grid &operator= (Grid &&other) noexcept

template<typename T1>
void copy (const Grid<T1, dim> &other)

template<typename T1>
void move (Grid<T1, dim> &&other) noexcept

template<typename Container>
inline void wrap (GridBase<T> &other, Container &&n)

template<typename Container>
inline void wrap (const GridBase<T> &other, Container &&n)

inline void wrap (Grid &other)

inline void wrap (const Grid &other)

```

Public Static Attributes

```
static constexpr UInt dimension = dim
```

Protected Attributes

```
std::array<UInt, dim> n
    shape of grid: size per dimension

std::array<UInt, dim + 1> strides
    strides for access

```

Private Types

```
template<typename ...D>

using is_valid_index = fold_trait<std::is_integral, D...>
```

Private Functions

```
template<typename Container>
void init (const Container &n, UInt nb_components)
    Init from standard container.

template<typename ...T1>
inline UInt unpackOffset (UInt offset, UInt index_pos, UInt index, T1... rest) const
    Unpacking the arguments of variadic ()

template<typename ...T1>
inline UInt unpackOffset (UInt offset, UInt index_pos, UInt index) const
    End case for recursion.

template<std::size_t tdim>
inline UInt computeOffset (std::array<UInt, tdim> tuple) const
    Computing offset for a tuple index.
```

```
template<typename T>
```

```
class GridBase
```

```
    #include <grid_base.hh> Dimension agnostic grid with components stored per points.
    Subclassed by tamaas::Grid< T, dim >
```

Public Types

```
using iterator = iterator_::iterator<T>

using const_iterator = iterator_::iterator<const T>

using value_type = T

using reference = value_type&
```

Public Functions

```
GridBase () = default
    Constructor by default.

inline GridBase (const GridBase &o)
    Copy constructor.
```

```

inline GridBase (GridBase &&o) noexcept
    Move constructor (transfers data ownership)

inline explicit GridBase (UInt nb_points, UInt nb_components = 1)
    Initialize with size.

virtual ~GridBase () = default
    Destructor.

inline virtual UInt getDimension () const
    Get grid dimension.

inline const T *getInternalData () const
    Get internal data pointer (const)

inline T *getInternalData ()
    Get internal data pointer (non-const)

inline UInt getNbComponents () const
    Get number of components.

inline void setNbComponents (UInt n)
    Set number of components.

inline virtual UInt dataSize () const
    Get total size.

inline UInt globalDataSize () const
    Get global data size.

inline UInt getNbPoints () const
    Get number of points.

inline UInt getGlobalNbPoints () const
    Get global number of points.

void uniformSetComponents (const GridBase<T> &vec)
    Set components.

inline void resize (UInt size)
    Resize.

inline void reserve (UInt size)
    Reserve storage w/o changing data logic.

inline virtual iterator begin (UInt n = 1)

inline virtual iterator end (UInt n = 1)

inline virtual const_iterator begin (UInt n = 1) const

inline virtual const_iterator end (UInt n = 1) const

inline decltype(auto) view () const

inline T &operator () (UInt i)

inline const T &operator () (UInt i) const

template<typename T1>

```

```
GridBase &operator+=(const GridBase<T1> &other)

template<typename T1>
GridBase &operator*=(const GridBase<T1> &other)

template<typename T1>
GridBase &operator--=(const GridBase<T1> &other)

template<typename T1>
GridBase &operator/=(const GridBase<T1> &other)

template<typename T1>
T dot (const GridBase<T1> &other) const

inline GridBase &operator+=(const T &e)

inline GridBase &operator*=(const T &e)

inline GridBase &operator--=(const T &e)

inline GridBase &operator/=(const T &e)

inline GridBase &operator=(const T &e)

template<typename DT, typename ST, UInt size>
GridBase &operator+=(const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase &operator--=(const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase &operator*=(const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase &operator/=(const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase &operator=(const StaticArray<DT, ST, size> &b)

inline T min () const

inline T max () const

inline T sum () const

inline T mean () const

inline T var () const

inline T l2norm () const

inline GridBase &operator=(const GridBase &o)

inline GridBase &operator=(GridBase &o)

inline GridBase &operator=(GridBase &&o) noexcept

template<typename T1>
inline void copy (const GridBase<T1> &other)

template<typename T1>
```

```

inline void move (GridBase<T1> &&other) noexcept

inline GridBase &wrap (GridBase &o)

inline GridBase &wrap (const GridBase &o)

template<typename T1>
inline GridBase<T> &operator+= (const GridBase<T1> &other)

template<typename T1>
inline GridBase<T> &operator-- (const GridBase<T1> &other)

template<typename T1>
inline GridBase<T> &operator*= (const GridBase<T1> &other)

template<typename T1>
inline GridBase<T> &operator/= (const GridBase<T1> &other)

template<typename DT, typename ST, UInt size>
GridBase<T> &operator+= (const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase<T> &operator-- (const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase<T> &operator*= (const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase<T> &operator/= (const StaticArray<DT, ST, size> &b)

template<typename DT, typename ST, UInt size>
GridBase<T> &operator= (const StaticArray<DT, ST, size> &b)

```

Protected Attributes

Array<*T*> **data**

UInt **nb_components** = 1

template<typename **T**, *UInt* **dim**>

class **GridHermitian** : public *tamaas::Grid*<*complex*<*T*>, *dim*>

#include <grid_hermitian.hh> Multi-dimensional, multi-component hermitian array.

This class represents an array of hermitian data, meaning it has dimensions of: $n_1 * n_2 * n_3 * \dots * (n_x / 2 + 1)$

However, it acts as a fully dimensioned array, returning a dummy reference for data outside its real dimension, allowing one to write full (and inefficient) loops without really worrying about the reduced dimension.

It would however be better to just use the true dimensions of the surface for all intents and purposes, as it saves computation time.

Public Types

```
using value_type = complex<T>
```

Public Functions

```
GridHermitian () = default
```

```
GridHermitian (const GridHermitian &o) = default
```

```
GridHermitian (GridHermitian &&o) noexcept = default
```

```
template<typename ...T1>  
inline complex<T> &operator () (T1... args)
```

```
template<typename ...T1>  
inline const complex<T> &operator () (T1... args) const
```

Public Static Functions

```
static inline std::array<UInt, dim> hermitianDimensions (std::array<UInt, dim> n)
```

```
template<typename Int, typename = std::enable_if_t<std::is_arithmetic<Int>::value>>  
static inline std::vector<Int> hermitianDimensions (std::vector<Int> n)
```

Private Functions

```
template<typename ...T1>  
inline void packTuple (UInt *t, UInt i, T1... args) const
```

```
template<typename ...T1>  
inline void packTuple (UInt *t, UInt i) const
```

```
template<template<typename, UInt> class Base, typename T, UInt base_dim, UInt dim>
```

```
class GridView : public Base<T, dim>
```

```
    #include <grid_view.hh> View type on grid This is a view on a contiguous chunk of data defined by a grid.
```

Public Types

```
using iterator = typename Base<T, dim>::iterator
```

```
using const_iterator = typename Base<T, dim>::const_iterator
```

```
using value_type = typename Base<T, dim>::value_type
```

```
using reference = typename Base<T, dim>::reference
```

Public Functions

GridView (*GridBase*<typename *Base*<*T*, *dim*>::value_type> &grid_base, const std::vector<*UInt*> &multi_index, *Int* component = -1)

Constructor.

inline **GridView** (*GridView* &&o) noexcept

Move constructor.

~GridView () override = default

Destructor.

inline *UInt* **dataSize** () const override

void **reserve** (*UInt* size) = delete

void **resize** (*UInt* size) = delete

inline *iterator* **begin** (*UInt* = 1) override

Iterators.

inline *iterator* **end** (*UInt* = 1) override

inline *const_iterator* **begin** (*UInt* = 1) const override

inline *const_iterator* **end** (*UInt* = 1) const override

Protected Attributes

Base<*T*, *base_dim*> ***grid**

template<typename **T**>

struct **helper**

#include <interface_impl.hh> Allocation helper for different float types.

template<>

struct **helper**<double>

#include <interface_impl.hh>

Public Types

using **complex** = fftw_complex

using **plan** = fftw_plan

Public Static Functions

```
static inline auto alloc_real (std::size_t size)

static inline auto alloc_complex (std::size_t size)
```

```
template<>
```

```
struct helper<float>
    #include <interface_impl.hh>
```

Public Types

```
using complex = fftwf_complex

using plan = fftwf_plan
```

Public Static Functions

```
static inline auto alloc_real (std::size_t size)

static inline auto alloc_complex (std::size_t size)
```

```
template<>
```

```
struct helper<long double>
    #include <interface_impl.hh>
```

Public Types

```
using complex = fftwl_complex

using plan = fftwl_plan
```

Public Static Functions

```
static inline auto alloc_real (std::size_t size)

static inline auto alloc_complex (std::size_t size)
```

```
template<model_type type>
```

```
class Hooke : public tamaas::IntegralOperator
    #include <hooke.hh> Applies Hooke's law of elasticity.
    Subclassed by tamaas::HookeField< type >
```

Public Functions

inline virtual *model_type* **getType** () const override

Type of underlying model.

inline virtual *IntegralOperator*::kind **getKind** () const override

Operator is local in real space.

inline virtual void **updateFromModel** () override

Does not update.

virtual void **apply** (*GridBase*<*Real*> &strain, *GridBase*<*Real*> &stress) const override

Apply *Hooke*'s tensor.

virtual std::pair<*UInt*, *UInt*> **matvecShape** () const override

LinearOperator shape.

virtual *GridBase*<*Real*> **matvec** (*GridBase*<*Real*> &X) const override

LinearOperator interface.

inline **IntegralOperator** (*Model* *model)

Constructor.

template<*model_type type*>

class **HookeField** : public *tamaas::Hooke*<*type*>

#include <hooke.hh>

Public Functions

HookeField (*Model* *model)

virtual void **apply** (*GridBase*<*Real*> &strain, *GridBase*<*Real*> &stress) const override

Apply *Hooke*'s tensor.

template<*UInt dim*>

struct **HookeFieldHelper**

Public Functions

inline void **operator** () (*MatrixProxy*<*Real*, *dim*, *dim*> sigma, *MatrixProxy*<const *Real*, *dim*, *dim*> epsilon,
const *Real* &mu, const *Real* &nu)

Public Members

influence::ElasticHelper<*dim*> **elasticity** = {0, 0}

class **IntegralOperator** : public *tamaas::FieldContainer*

#include <integral_operator.hh> Generic class for integral operators.

Subclassed by *tamaas::detail::ComputeOperator*< *Compute_t* >, *tamaas::Hooke*< *type* >, *tamaas::VolumePotential*<
type >, *tamaas::Westergaard*< *mtype*, *otype* >, *tamaas::Westergaard*< *model_type::basic_2d*, *IntegralOperator::neu-*
mann >

Public Types

enum **kind**

Kind of operator.

Values:

enumerator **neumann**

enumerator **dirichlet**

enumerator **dirac**

Public Functions

inline **IntegralOperator** (*Model* *model)

Constructor.

virtual void **apply** (*GridBase<Real>* &input, *GridBase<Real>* &output) const = 0

Apply operator on input.

inline virtual void **applyIf** (*GridBase<Real>* &input, *GridBase<Real>* &output, const
std::function<bool(*UInt*)>&) const

Apply operator on filtered input.

inline virtual void **updateFromModel** ()

Update any data dependent on model parameters.

inline const *Model* &**getModel** () const

Get model.

inline virtual *kind* **getKind** () const

Kind.

virtual *model_type* **getType** () const

Type.

inline virtual *Real* **getOperatorNorm** ()

Norm.

inline virtual std::pair<*UInt*, *UInt*> **matvecShape** () const

Dense shape (for Scipy integration)

inline virtual *GridBase<Real>* **matvec** (*GridBase<Real>* &) const

matvec definition

Protected Attributes

Model *model = nullptr

template<U**Int** interpolation_order>

class **Integrator**

#include <integrator.hh>

Public Static Functions

template<bool upper, U**Int** shape>

static inline *Real* **G0** (*Real* q, *Real* r, *Real* xc)

Standard integral of $\exp(\pm qy)\phi(y)$ over an element of radius r and center x_c

template<bool upper, U**Int** shape>

static inline *Real* **G1** (*Real* q, *Real* r, *Real* xc)

Standard integral of $qy \exp(\pm qy)\phi(y)$ over an element of radius r and center x_c

template<U**Int** shape>

static inline constexpr *Real* **F** (*Real* r)

Standard integral of $\phi(y)$ over an element of radius r and center x_c . Used for fundamental mode

Private Types

using **element** = *ExponentialElement*<interpolation_order>

Private Static Attributes

static constexpr std::pair<*Real*, *Real*> **bounds** = {-1, 1}

template<typename **T**, U**Int** dim>

struct **Internal**

#include <internal.hh>

Public Functions

inline void **initialize** ()

Initialize previous field.

inline void **update** ()

Update the field previous state.

inline void **reset** ()

Reset the current field.

inline void **operator=** (decltype(*field*) &&field)

Assign the field pointer.

```
inline decltype(field) ::element_type & operator* ()
```

Dereference the field pointer.

```
inline const decltype(field) ::element_type & operator* () const
```

Dereference the field pointer (const version)

```
inline operator std::shared_ptr<GridBase<T>> ()
```

Upcast to *GridBase* pointer.

Public Members

```
std::shared_ptr<Grid<T, dim>> field
```

Stored field.

```
decltype(field) previous
```

```
template<typename T>
```

```
struct is_arithmetic : public std::is_arithmetic<T>
```

```
    #include <static_types.hh>
```

```
template<typename T>
```

```
struct is_arithmetic<thrust::complex<T>> : public true_type
```

```
    #include <static_types.hh>
```

```
template<typename T>
```

```
struct is_policy : public false_type
```

```
    #include <loop.hh>
```

```
template<>
```

```
struct is_policy<const thrust::detail::device_t&> : public true_type
```

```
    #include <loop.hh>
```

```
template<>
```

```
struct is_policy<const thrust::detail::device_t> : public true_type
```

```
    #include <loop.hh>
```

```
template<>
```

```
struct is_policy<const thrust::detail::host_t&> : public true_type
```

```
    #include <loop.hh>
```

```
template<>
```

```
struct is_policy<const thrust::detail::host_t> : public true_type
```

```
    #include <loop.hh>
```

```
template<>
```

```

struct is_policy<thrust::detail::device_t> : public true_type
    #include <loop.hh>

template<>

struct is_policy<thrust::detail::host_t> : public true_type
    #include <loop.hh>

template<class Type>

struct is_proxy : public false_type
    #include <static_types.hh>

template<typename T, UInt n, UInt m>

struct is_proxy<MatrixProxy<T, n, m>> : public true_type
    #include <static_types.hh>

template<typename T, UInt n>

struct is_proxy<SymMatrixProxy<T, n>> : public true_type
    #include <static_types.hh>

template<template<typename, typename, UInt...> class StaticParent, typename T, UInt... dims>

struct is_proxy<TensorProxy<StaticParent, T, dims...>> : public true_type
    #include <static_types.hh>

template<typename T, UInt n>

struct is_proxy<VectorProxy<T, n>> : public true_type
    #include <static_types.hh>

template<typename Container>

struct is_valid_container : public std::is_same<std::remove_cv_t<std::decay_t<Container>::value_type>,
std::remove_cv_t<ValueType>>
    #include <ranges.hh>

template<UInt dim>

class Isopowerlaw : public tamaas::Filter<dim>
    #include <isopowerlaw.hh> Class representing an isotropic power law spectrum.

```

Public Functions

```

virtual void computeFilter (GridHermitian<Real, dim> &filter_coefficients) const override
    Compute filter coefficients.

inline Real operator () (const VectorProxy<Real, dim> &q_vec) const
    Compute a point of the PSD.

Real rmsHeights () const
    Analytical rms of heights.

```

`std::vector<Real> moments ()` const
Analytical moments.

`Real alpha ()` const
Analytical Nayak's parameter.

`Real rmsSlopes ()` const
Analytical RMS slopes.

`Real radialPSDMoment (Real q)` const
Computes $\int k^q \phi(k) k dk$ from 0 to ∞

`Real elasticEnergy ()` const
Compute full contact energy (unscaled by E^*)

`TAMAAS_ACCESSOR (q0, UInt, Q0)`

`TAMAAS_ACCESSOR (q1, UInt, Q1)`

`TAMAAS_ACCESSOR (q2, UInt, Q2)`

`TAMAAS_ACCESSOR (hurst, Real, Hurst)`

`Real radialPSDMoment (Real q)` const

`Real radialPSDMoment (Real)` const

Protected Attributes

`UInt q0`

`UInt q1`

`UInt q2`

`Real hurst`

```
class IsotropicHardening : public tamaas::Material
#include <isotropic_hardening.hh>
```

Public Functions

`IsotropicHardening (Model *model, Real sigma_0, Real h)`
Constructor.

void `computeInelasticDeformationIncrement` (`Field` &increment, const `Field` &strain, const `Field` &strain_increment)

Compute plastic strain increment with radial return algorithm.

void `computeStress` (`Field` &stress, const `Field` &strain, const `Field` &strain_increment) override
Compute stress.

```

void computeEigenStress (Field &stress, const Field &strain, const Field &strain_increment) override
    Compute stress due to plastic strain increment.

virtual void update () override
    Update internal variables.

void applyTangent (Field &output, const Field &input, const Field &strain, const Field &strain_increment)
    override

inline Real getHardeningModulus () const

inline Real getYieldStress () const

inline const GridBase<Real> &getPlasticStrain () const

inline GridBase<Real> &getPlasticStrain ()

inline void setHardeningModulus (Real h_)

inline void setYieldStress (Real sigma_0_)

```

Public Static Functions

```

static inline Real hardening (Real p, Real h, Real sigma_0)
    Linear hardening function.

```

Protected Attributes

Real **sigma_0**

Real **h**
 < initial yield stress

Internal<*Real*, *dim*> **plastic_strain**
 < hardening modulus

Internal<*Real*, *dim*> **cumulated_plastic_strain**

Private Types

using **parent** = *Material*

using **trait** = *model_type_traits*<*type*>

using **Field** = typename *parent*::*Field*

using **Mat** = *SymMatrixProxy*<*Real*, *dim*>

using **CMat** = *SymMatrixProxy*<const *Real*, *dim*>

Private Static Attributes

```
static constexpr auto type = model_type::volume_2d
```

```
static constexpr UInt dim = trait::dimension
```

```
template<typename T>
```

```
class iterator
```

```
    #include <iterator.hh> Subclassed by tamaas::iterator_::iterator_view< T >
```

Public Types

```
using value_type = T
```

```
using difference_type = std::ptrdiff_t
```

```
using pointer = T*
```

```
using reference = T&
```

```
using iterator_category = thrust::random_access_device_iterator_tag
```

Public Functions

```
inline iterator (pointer data, difference_type step_size)  
    constructor
```

```
inline iterator (const iterator &o)  
    copy constructor
```

```
inline iterator (iterator &&o) noexcept  
    move constructor
```

```
template<typename U, typename = std::enable_if_t<std::is_convertible<T, U>::value>>  
inline iterator (const iterator<U> &o)  
    copy from a different qualified type
```

```
template<typename U, typename = std::enable_if_t<std::is_convertible<T, U>::value>>  
inline iterator (iterator<U> &&o)  
    move from a different qualified type
```

```
inline iterator &operator= (const iterator &o)
```

```
inline iterator &operator= (iterator &&o) noexcept
```

```
inline reference operator* ()  
    dereference iterator
```

```
inline reference operator* () const
    dereference iterator

inline iterator &operator+=(difference_type a)
    increment with given offset

inline iterator &operator++ ()
    increment iterator

inline bool operator< (const iterator &a) const
    comparison

inline bool operator!= (const iterator &a) const
    inequality

inline bool operator== (const iterator &a) const
    equality

inline difference_type operator- (const iterator &a) const
    needed for OpenMP range calculations

inline void setStep (difference_type s)
```

Protected Attributes

T ***data**

difference_type **step**

Friends

friend class iterator

```
class iterator : public tamaas::iterator_::iterator<ValueType>
    #include <ranges.hh>
```

Public Types

using **value_type** = LocalType

using **reference** = *value_type*

Public Functions

```
inline iterator (const parent &o)

inline reference operator* ()

inline reference operator* () const
```

Private Types

```
using parent = iterator_::iterator<ValueType>

template<direction dir>

struct iterator

    #include <accumulator.hh> Forward/Backward iterator for integration passes.
```

Public Types

```
using integ = Integrator<1>
```

Public Functions

```
inline iterator (Accumulator &acc, Int k)
    Constructor.

inline iterator (const iterator &o)
    Copy constructor.

inline bool operator!= (const iterator &o) const
    Compare.

inline iterator &operator++ ()
    Increment.

inline std::tuple<Int, Real, BufferType&, BufferType&> operator* ()
    Derefence iterator (TODO uniformize tuple types)
```

Public Static Attributes

```
static constexpr bool upper = dir == direction::backward
```

Protected Functions

```
inline bool setup ()  
    Update index layer and element info.  
  
inline void next ()  
    Set current layer and update element index.
```

Protected Attributes

Accumulator &**acc**

Int **k** = 0
 accumulator holder
 element index

Int **l** = 0
 layer index

Real **r** = 0
 element radius

Real **xc** = 0
 element center

Protected Static Functions

```
static inline Int layer (Int element)  
    Element index => layer index.  
  
static inline Int nextElement (Int element)  
    Next element index in right direction.
```

```
template<typename T>  
class iterator_view : private tamaas::iterator_::iterator<T>  
    #include <iterator.hh>
```

Public Types

```
using value_type = T  
  
using difference_type = std::ptrdiff_t  
  
using pointer = T*  
  
using reference = T&
```

Public Functions

inline **iterator_view** (*pointer* data, std::size_t start, ptrdiff_t offset, std::vector<*UInt*> strides, std::vector<*UInt*> sizes)

Constructor.

inline **iterator_view** (const *iterator_view* &o)

inline *iterator_view* &**operator=** (const *iterator_view* &o)

Protected Functions

inline void **computeAccessOffset** ()

Protected Attributes

std::vector<*UInt*> **strides**

std::vector<*UInt*> **n**

std::vector<*UInt*> **tuple**

class **Kato** : public *tamaas::ContactSolver*

#include <kato.hh> Subclassed by *tamaas::BeckTeboulle*, *tamaas::Condat*, *tamaas::PolonskyKeerTan*

Public Functions

Kato (*Model* &model, const *GridBase<Real>* &surface, *Real* tolerance, *Real* mu)

Constructor.

Real **solve** (*GridBase<Real>* &p0, *UInt* proj_iter)

Solve.

Real **solveRelaxed** (*GridBase<Real>* &g0)

Solve relaxed problem.

Real **solveRegularized** (*GridBase<Real>* &p0, *Real* r)

Solve regularized problem.

Real **computeCost** (bool use_tresca = false)

Compute cost function.

Compute mean of the field taking each component separately.

template<*model_type* **type**>

Real **solveTpl** (*GridBase<Real>* &p0, *UInt* proj_iter)

Template for solve function.

template<*model_type* **type**>

Real **solveRelaxedTmp1** (*GridBase<Real>* &g0)
 Template for solveRelaxed function.

template<*model_type* type>
Real **solveRegularizedTmp1** (*GridBase<Real>* &p0, *Real* r)
 Template for solveRegularized function.

template<*model_type* type>
 void **initSurfaceWithComponents** ()
 Creates surfaceComp from surface.

template<*UInt* comp>
 void **computeGradient** (bool use_tresca = false)
 Compute gradient of functional.

template<*UInt* comp>
 void **enforcePressureConstraints** (*GridBase<Real>* &p0, *UInt* proj_iter)
 Project pressure on friction cone.
 Projects $\vec{p}$ on $\mathcal{C}$ and $\mathcal{D}$. Projects $\vec{p}$ on $\mathcal{C}$ and $\mathcal{D}$.

template<*UInt* comp>
 void **enforcePressureMean** (*GridBase<Real>* &p0)
 Project on C.

template<*UInt* comp>
 void **enforcePressureCoulomb** ()
 Project on D.

template<*UInt* comp>
 void **enforcePressureTresca** ()
 Project on D (Tresca)

template<*UInt* comp>
Vector<Real, comp> **computeMean** (*GridBase<Real>* &field)
 Compute mean value of field.
 Compute mean of the field taking each component separately.

template<*UInt* comp>
 void **addUniform** (*GridBase<Real>* &field, *GridBase<Real>* &vec)
 Add vector to each point of field.

template<*model_type* type>
 void **computeValuesForCost** (*GridBase<Real>* &lambda, *GridBase<Real>* &eta, *GridBase<Real>* &p_N,
GridBase<Real> &p_C)
 Compute grids of dual and primal variables.

template<*model_type* type>
 void **computeValuesForCostTresca** (*GridBase<Real>* &lambda, *GridBase<Real>* &eta,
GridBase<Real> &p_N, *GridBase<Real>* &p_C)
 Compute dual and primal variables with Tresca friction.

template<*UInt* comp>
 void **computeFinalGap** ()
 Compute total displacement.

Public Static Functions

static *Real* **regularize** (*Real* x, *Real* r)

Regularization function with factor r (0 -> unregugularized)

Protected Attributes

BEEngine &**engine**

GridBase<*Real*> ***gap** = nullptr

GridBase<*Real*> ***pressure** = nullptr

std::unique_ptr<*GridBase*<*Real*>> **surfaceComp** = nullptr

Real **mu** = 0

UInt **N** = 0

class **KatoSaturated** : public *tamaas::PolonskyKeerRey*

#include <kato_saturated.hh> Polonsky-Keer algorithm with saturation of the pressure.

Public Functions

KatoSaturated (*Model* &model, const *GridBase*<*Real*> &surface, *Real* tolerance, *Real* pmax)

Constructor.

~KatoSaturated () override = default

virtual *Real* **solve** (std::vector<*Real*> load) override

Solve.

virtual *Real* **meanOnUnsaturated** (const *GridBase*<*Real*> &field) const override

Mean on unsaturated constraint zone.

virtual *Real* **computeSquaredNorm** (const *GridBase*<*Real*> &field) const override

Compute squared norm.

virtual void **updateSearchDirection** (*Real* factor) override

Update search direction.

virtual *Real* **computeCriticalStep** (*Real* target = 0) override

Compute critical step.

virtual bool **updatePrimal** (*Real* step) override

Update primal and check non-admissible state.

virtual *Real* **computeError** () const override

Compute error/stopping criterion.

virtual void **enforceMeanValue** (*Real* mean) override

Enfore mean value constraint.

virtual void **enforceAdmissibleState** () override

Enforce contact constraints on final state.

inline *Real* **getPMax** () const

Access to pmax.

inline void **setPMax** (*Real* p)

Set pax.

Protected Attributes

Real **pmax** = std::numeric_limits<*Real*>::max()

saturation pressure

template<*UInt* **dim**, *UInt* **derivative_order**>

class **Kelvin**

#include <influence.hh> Class for the *Kelvin* tensor.

template<*model_type* **type**, *UInt* **derivative**>

class **Kelvin** : public *tamaas::VolumePotential*<*type*>

#include <kelvin.hh> *Kelvin* tensor.

Subclassed by *tamaas::Mindlin*<*type*, *derivative*>

Public Functions

Kelvin (*Model* *model)

Constructor.

void **applyIf** (*GridBase*<*Real*> &source, *GridBase*<*Real*> &out, *filter_t* pred) const override

Apply the Kelvin-tensor_order operator.

void **setIntegrationMethod** (*integration_method* method, *Real* cutoff)

Set the integration method for volume operator.

virtual std::pair<*UInt*, *UInt*> **matvecShape** () const override

Dense shape (for Scipy integration)

virtual *GridBase*<*Real*> **matvec** (*GridBase*<*Real*> &X) const override

matvec definition

Protected Types

```
using KelvinInfluence = influence::Kelvin<trait::dimension, derivative>
```

```
using ktrait = detail::KelvinTrait<KelvinInfluence>
```

```
using Source = typename ktrait::source_t
```

```
using Out = typename ktrait::out_t
```

```
using filter_t = typename parent::filter_t
```

Protected Attributes

```
integration_method method = integration_method::linear
```

```
Real cutoff
```

Private Types

```
using trait = model_type_traits<type>
```

```
using dttrait = derivative_traits<derivative>
```

```
using parent = VolumePotential<type>
```

Private Functions

```
void linearIntegral (GridBase<Real> &out, KelvinInfluence &kelvin) const
```

```
void cutoffIntegral (GridBase<Real> &out, KelvinInfluence &kelvin) const
```

```
template<>
```

```
class Kelvin<3, 0>
```

```
    #include <influence.hh> Kelvin base tensor See arXiv:1811.11558 for details.
```

```
    Subclassed by tamaas::influence::Kelvin<3, 1>
```

Public Functions

```
inline Kelvin (Real mu, Real nu)
```

```
template<bool upper, bool apply_q_power = false, typename ST>
inline Vector<Complex, dim> applyU0 (const VectorProxy<const Real, dim - 1> &q, const
                                     StaticVector<Complex, ST, dim> &f) const
```

```
template<bool upper, bool apply_q_power = false, typename ST>
inline Vector<Complex, dim> applyU1 (const VectorProxy<const Real, dim - 1> &q, const
                                     StaticVector<Complex, ST, dim> &f) const
```

Protected Attributes

```
const Real mu
```

```
const Real b
```

Protected Static Attributes

```
static constexpr UInt dim = 3
```

```
static constexpr UInt order = 0
```

```
template<>
```

```
class Kelvin<3, 1> : protected tamaas::influence::Kelvin<3, 0>
```

```
    #include <influence.hh> Kelvin first derivative.
```

```
    Subclassed by tamaas::influence::Kelvin< 3, 2 >
```

Public Functions

```
template<bool upper, bool apply_q_power = false, typename ST>
inline Vector<Complex, dim> applyU0 (const VectorProxy<const Real, dim - 1> &q, const
                                     StaticMatrix<Complex, ST, dim, dim> &f) const
```

```
template<bool upper, bool apply_q_power = false, typename ST>
inline Vector<Complex, dim> applyU1 (const VectorProxy<const Real, dim - 1> &q, const
                                     StaticMatrix<Complex, ST, dim, dim> &f) const
```

Protected Static Attributes

```
static constexpr UInt dim = parent::dim
```

```
static constexpr UInt order = parent::order + 1
```

Private Types

```
using parent = Kelvin<3, 0>
```

```
template<>
```

```
class Kelvin<3, 2> : protected tamaas::influence::Kelvin<3, 1>
```

```
    #include <influence.hh> Kelvin second derivative.
```

Public Functions

```
template<bool upper, bool apply_q_power = false, typename ST>  
inline Matrix<Complex, dim, dim> applyU0 (const VectorProxy<const Real, dim - 1> &q, const  
                                             StaticMatrix<Complex, ST, dim, dim> &f) const
```

```
template<bool upper, bool apply_q_power = false, typename ST>  
inline Matrix<Complex, dim, dim> applyU1 (const VectorProxy<const Real, dim - 1> &q, const  
                                             StaticMatrix<Complex, ST, dim, dim> &f) const
```

```
template<typename ST>  
inline Matrix<Complex, dim, dim> applyDiscontinuityTerm (const StaticMatrix<Complex, ST, dim,  
                                                             dim> &f) const
```

Private Types

```
using parent = Kelvin<3, 1>
```

Private Static Attributes

```
static constexpr UInt dim = parent::dim
```

```
static constexpr UInt order = parent::order + 1
```

```
template<model_type type, typename kelvin_t, typename = typename KelvinTrait<kelvin_t>::source_t>
```

```
struct KelvinHelper
```

```
    #include <kelvin_helper.hh> Helper to apply integral representation on output.
```

Public Types

```
using trait = model_type_traits<type>
```

```
using BufferType = GridHermitian<Real, bdim>
```

```
using source_t = typename KelvinTrait<kelvin_t>::source_t
```

```
using out_t = typename KelvinTrait<kelvin_t>::out_t
```

Public Functions

```
virtual ~KelvinHelper () = default
```

```
inline void applyIntegral (std::vector<BufferType> &source, std::vector<BufferType> &out, const  
                        Grid<Real, bdim> &wavevectors, Real domain_size, const kelvin_t &kelvin)
```

Apply the regular part of *Kelvin* to source and sum into output.

This function performs the linear integration algorithm using the accumulator.

```
inline void applyIntegral (std::vector<BufferType> &source, BufferType &out, UInt layer, const Grid<Real,  
                        bdim> &wavevectors, Real domain_size, Real cutoff, const kelvin_t &kelvin)
```

Apply the regular part of *Kelvin* to source and sum into output.

This function performs the cutoff integration algorithm. Not to be confused with its overload above.

```
inline void applyFreeTerm (std::vector<BufferType> &source, std::vector<BufferType> &out, const kelvin_t  
                        &kelvin)
```

Apply free term of distribution derivative.

```
inline void applyFreeTerm (BufferType&, BufferType&, const kelvin_t&)
```

Apply free term of distribution derivative (layer)

```
inline void makeFundamentalGreatAgain (std::vector<BufferType> &out)
```

Making the output free of communist NaN.

```
inline void makeFundamentalGreatAgain (BufferType&)
```

Making the output free of communist NaN (layer)

```
inline void applyFreeTerm (BufferType &source, BufferType &out, const influence::Kelvin<3, 2> &kelvin)
```

Applying free term for double gradient of *Kelvin*.

Public Static Attributes

```
static constexpr UInt dim = trait::dimension
```

```
static constexpr UInt bdim = trait::boundary_dimension
```

Protected Attributes

Accumulator<*type*, *source_t*> **accumulator**

template<typename **T**>

struct **KelvinTrait**

#include <kelvin_helper.hh> Trait for kelvin local types.

template<*UInt dim*>

struct **KelvinTrait**<*influence::Boussinesq*<*dim*, 0>>

#include <kelvin_helper.hh>

Public Types

using **source_t** = *VectorProxy*<*Complex*, *dim*>

using **out_t** = *VectorProxy*<*Complex*, *dim*>

template<*UInt dim*>

struct **KelvinTrait**<*influence::Boussinesq*<*dim*, 1>>

#include <kelvin_helper.hh>

Public Types

using **source_t** = *VectorProxy*<*Complex*, *dim*>

using **out_t** = *SymMatrixProxy*<*Complex*, *dim*>

template<*UInt dim*>

struct **KelvinTrait**<*influence::Kelvin*<*dim*, 0>>

#include <kelvin_helper.hh>

Public Types

using **source_t** = *VectorProxy*<*Complex*, *dim*>

using **out_t** = *VectorProxy*<*Complex*, *dim*>

template<*UInt dim*>

struct **KelvinTrait**<*influence::Kelvin*<*dim*, 1>>

#include <kelvin_helper.hh>

Public Types

```
using source_t = SymMatrixProxy<Complex, dim>

using out_t = VectorProxy<Complex, dim>

template<UInt dim>

struct KelvinTrait<influence::Kelvin<dim, 2>>
    #include <kelvin_helper.hh>
```

Public Types

```
using source_t = SymMatrixProxy<Complex, dim>

using out_t = SymMatrixProxy<Complex, dim>
```

class **Logger**

#include <logger.hh> Logging class Inspired from <https://www.drdobbs.com/cpp/logging-in-c/201804215> by Petru Marginean.

Public Functions

```
~Logger () noexcept
    Writing to stderr.

std::ostream &get (LogLevel level = LogLevel::debug)
    Get stream.

inline decltype(auto) getWishLevel () const
    Get wish log level.
```

Public Static Functions

```
static inline decltype(auto) getCurrentLevel ()
    Get current log level.

static void setLevel (LogLevel level)
    Set acceptable logging level.
```

Private Members

`std::ostringstream stream`

`LogLevel wish_level = LogLevel::debug`

Private Static Attributes

`static LogLevel current_level = LogLevel::info`

class Loop

#include <loop.hh> Singleton class for automated loops using lambdas.

This class is sweet candy :) It provides abstraction of the paralelism paradigm used in loops and allows simple and less error-prone loop syntax, with minimum boiler plate. I love it <3

Public Functions

`Loop () = delete`

Constructor.

Public Static Functions

`template<typename T>`
`static inline arange<T> range ( T size )`

`template<typename T, typename U>`
`static inline arange<T> range ( U start, T size )`

`template<typename Functor, typename ...Ranges>`
`static inline auto loop ( Functor &&func, Ranges&&... ranges ) -> typename std::enable_if<not`
`is_policy<Functor>::value, void>::type`

Loop functor over ranges.

`template<typename DerivedPolicy, typename Functor, typename ...Ranges>`
`static inline void loop ( const thrust::execution_policy<DerivedPolicy> &policy, Functor &&func, Ranges&&... ranges )`

Loop over ranges with non-default policy.

`template<operation op = operation::plus, typename Functor, typename ...Ranges>`
`static inline auto reduce ( Functor &&func, Ranges&&... ranges ) -> typename std::enable_if<not`
`is_policy<Functor>::value,`
`decltype(func(std::declval<reference_type<Ranges>>()...))>::type`

Reduce functor over ranges.

`template<operation op = operation::plus, typename DerivedPolicy, typename Functor, typename ...Ranges>`
`static inline auto reduce ( const thrust::execution_policy<DerivedPolicy> &policy, Functor &&func,`
`Ranges&&... ranges ) -> decltype(func(std::declval<reference_type<Ranges>>()...))`

Reduce over ranges with non-default policy.

Private Types

```
template<typename T>
```

```
using reference_type = typename std::decay<T>::type::reference
```

Private Static Functions

```
template<typename DerivedPolicy, typename Functor, typename ...Ranges>
```

```
static void loopImpl (const thrust::execution_policy<DerivedPolicy> &policy, Functor &&func, Ranges&&...  
                      ranges)
```

Loop over ranges and apply functor.

```
template<operation op, typename DerivedPolicy, typename Functor, typename ...Ranges>
```

```
static auto reduceImpl (const thrust::execution_policy<DerivedPolicy> &policy, Functor &&func,  
                        Ranges&&... ranges) -> decltype(func(std::declval<reference_type<Ranges>>()...))
```

Loop over ranges, apply functor and reduce result.

```
class Material
```

```
#include <material.hh> Subclassed by tamaas::IsotropicHardening, tamaas::MFrontMaterial
```

Public Functions

```
inline Material (Model *model)
```

Constructor.

```
virtual ~Material () = default
```

Destructor.

```
virtual void computeStress (Field &stress, const Field &strain, const Field &strain_increment) = 0
```

Compute the stress from total strain and strain increment.

```
virtual void computeEigenStress (Field &stress, const Field &strain, const Field &strain_increment) = 0
```

Compute stress due to inelastic increment.

```
virtual void update () = 0
```

Update internal variables.

```
inline virtual void applyTangent (Field&, const Field&, const Field&, const Field&)
```

Appl't consistent tangent.

Protected Types

```
using Field = GridBase<Real>
```

Protected Attributes

Model *model

```
class MaugisAdhesionFunctional : public tamaas::functional::AdhesionFunctional  
    #include <adhesion_functional.hh> Constant adhesion functional.
```

Public Functions

```
inline MaugisAdhesionFunctional (const GridBase<Real> &surface)
```

Explicit declaration of constructor for swig.

```
virtual Real computeF (GridBase<Real> &gap, GridBase<Real> &pressure) const override
```

Compute the total adhesion energy.

```
virtual void computeGradF (GridBase<Real> &gap, GridBase<Real> &gradient) const override
```

Compute the gradient of the adhesion functional.

```
class MetaFunctional : public tamaas::functional::Functional  
    #include <meta_functional.hh> Meta functional that contains list of functionals.
```

Public Functions

```
virtual Real computeF (GridBase<Real> &variable, GridBase<Real> &dual) const override
```

Compute functional.

```
virtual void computeGradF (GridBase<Real> &variable, GridBase<Real> &gradient) const override
```

Compute functional gradient.

```
void addFunctionalTerm (std::shared_ptr<Functional> functional)
```

Add functional to the list.

```
void clear ()
```

Clears the functional list.

Protected Attributes

FunctionalList functionals

Private Types

```
using FunctionalList = std::list<std::shared_ptr<Functional>>
```

```
class MFrontMaterial : public tamaas::Material  
    #include <mfront_material.hh>
```

Public Functions

MFrontMaterial (*Model* *model, std::string material)

Constructor.

Private Types

using **parent** = *Material*

using **trait** = *model_type_traits*<*type*>

using **Field** = typename *parent*::Field

Private Static Attributes

static constexpr auto **type** = *model_type::volume_2d*

static constexpr *UInt* **dim** = *trait*::dimension

template<*model_type* **type**, *UInt* **derivative**>

class **Mindlin** : public *tamaas::Kelvin*<*type*, *derivative*>

#include <mindlin.hh> *Mindlin* tensor.

Public Functions

Mindlin (*Model* *model)

Constructor.

void **applyIf** (*GridBase*<*Real*> &source, *GridBase*<*Real*> &out, *filter_t* pred) const override

Apply the Mindlin-tensor_order operator.

Protected Functions

void **linearIntegral** (*GridBase*<*Real*> &out) const

void **cutoffIntegral** (*GridBase*<*Real*> &out) const

Protected Attributes

mutable *GridHermitian*<*Real*, *trait*::boundary_dimension> **surface_tractions**

Private Types

using **trait** = *model_type_traits*<*type*>

using **parent** = *Kelvin*<*type*, *derivative*>

using **filter_t** = typename *parent*::filter_t

class **Model** : public *tamaas::FieldContainer*

#include <model.hh> *Model* containing pressure and displacement This class is a container for the model fields. It is supposed to be dimension agnostic, hence the *GridBase* members.

Subclassed by *tamaas::ModelTemplate*< *type* >

Public Functions

void **setElasticity** (*Real* E, *Real* nu)

Set elasticity parameters.

inline *Real* **getHertzModulus** () const

Get Hertz contact modulus.

inline *Real* **getYoungModulus** () const

Get Young's modulus.

inline *Real* **getPoissonRatio** () const

Get Poisson's ratio.

inline *Real* **getShearModulus** () const

Get shear modulus.

inline void **setYoungModulus** (*Real* E_)

Set Young's modulus.

inline void **setPoissonRatio** (*Real* nu_)

Set Poisson's ratio.

virtual *model_type* **getType** () const = 0

Get model type.

const std::vector<*Real*> &**getSystemSize** () const

Get system physical size.

virtual std::vector<*Real*> **getBoundarySystemSize** () const = 0

Get boundary system physical size.

```

const std::vector<UInt> &getDiscretization () const
    Get discretization.

virtual std::vector<UInt> getGlobalDiscretization () const = 0
    Get discretization of global MPI system.

virtual std::vector<UInt> getBoundaryDiscretization () const = 0
    Get boundary discretization.

inline BEEngine &getBEEngine ()
    Get boundary element engine.

void applyElasticity (GridBase<Real> &stress, const GridBase<Real> &strain) const
    Apply Hooke's law.

void solveNeumann ()
    Solve Neumann problem using default neumann operator.

void solveDirichlet ()
    Solve Dirichlet problem using default dirichlet operator.

template<typename Operator>
std::shared_ptr<IntegralOperator> registerIntegralOperator (const std::string &name)
    Register a new integral operator.

void registerIntegralOperator (const std::string &name, std::shared_ptr<IntegralOperator> op)
    Register external operator.

std::shared_ptr<IntegralOperator> getIntegralOperator (const std::string &name) const
    Get a registered integral operator.

std::vector<std::string> getIntegralOperators () const
    Get list of integral operators.

inline const auto &getIntegralOperatorsMap () const
    Get operators mapcar.

void updateOperators ()
    Tell operators to update their cache.

virtual void setIntegrationMethod (integration_method method, Real cutoff) = 0
    Set integration method for registered volume operators.

GridBase<Real> &getTraction ()
    Get pressure.

const GridBase<Real> &getTraction () const
    Get pressure.

GridBase<Real> &getDisplacement ()
    Get displacement.

const GridBase<Real> &getDisplacement () const
    Get displacement.

template<class T>
inline bool isBoundaryField (const GridBase<T> &field) const
    Determine if a field is defined on boundary.

```

```
std::vector<std::string> getBoundaryFields () const
    Return list of fields defined on boundary.

void addDumper (std::shared_ptr<ModelDumper> dumper)
    Set the dumper object.

void dump () const
    Dump the model.
```

Protected Functions

```
inline Model (std::vector<Real> system_size, std::vector<UInt> discretization)
    Constructor.
```

Protected Attributes

```
Real E = 1

Real nu = 0

std::vector<Real> system_size

std::vector<UInt> discretization

std::unique_ptr<BEngine> engine = nullptr

std::unordered_map<std::string, std::shared_ptr<IntegralOperator>> operators

std::vector<std::shared_ptr<ModelDumper>> dumpers
```

Friends

```
friend std::ostream &operator<< (std::ostream &o, const Model &_this)

class model_type_error : public domain_error
    #include <errors.hh>

template<model_type type>

struct model_type_traits
    #include <model_type.hh> Trait class to store physical dimension of domain, of boundary and number of components

template<>

struct model_type_traits<model_type::basic_1d>
    #include <model_type.hh>
```

Public Static Attributes

```
static constexpr char repr[] = {"basic_1d"}
```

```
static constexpr UInt dimension = 1
```

```
static constexpr UInt components = 1
```

```
static constexpr UInt boundary_dimension = 1
```

```
static constexpr UInt voigt = voigt_size<1>::value
```

```
static const std::vector<UInt> indices = { }
```

```
template<>
```

```
struct model_type_traits<model_type::basic_2d>
```

```
    #include <model_type.hh>
```

Public Static Attributes

```
static constexpr char repr[] = {"basic_2d"}
```

```
static constexpr UInt dimension = 2
```

```
static constexpr UInt components = 1
```

```
static constexpr UInt boundary_dimension = 2
```

```
static constexpr UInt voigt = voigt_size<1>::value
```

```
static const std::vector<UInt> indices = { }
```

```
template<>
```

```
struct model_type_traits<model_type::surface_1d>
```

```
    #include <model_type.hh>
```

Public Static Attributes

```
static constexpr char repr[] = {"surface_1d"}

static constexpr UInt dimension = 1

static constexpr UInt components = 2

static constexpr UInt boundary_dimension = 1

static constexpr UInt voigt = voigt_size<2>::value

static const std::vector<UInt> indices = {}

template<>
struct model_type_traits<model_type::surface_2d>
    #include <model_type.hh>
```

Public Static Attributes

```
static constexpr char repr[] = {"surface_2d"}

static constexpr UInt dimension = 2

static constexpr UInt components = 3

static constexpr UInt boundary_dimension = 2

static constexpr UInt voigt = voigt_size<3>::value

static const std::vector<UInt> indices = {}

template<>
struct model_type_traits<model_type::volume_1d>
    #include <model_type.hh>
```

Public Static Attributes

```
static constexpr char repr[] = {"volume_1d"}
```

```
static constexpr UInt dimension = 2
```

```
static constexpr UInt components = 2
```

```
static constexpr UInt boundary_dimension = 1
```

```
static constexpr UInt voigt = voigt_size<2>::value
```

```
static const std::vector<UInt> indices = {0}
```

```
template<>
```

```
struct model_type_traits<model_type::volume_2d>
    #include <model_type.hh>
```

Public Static Attributes

```
static constexpr char repr[] = {"volume_2d"}
```

```
static constexpr UInt dimension = 3
```

```
static constexpr UInt components = 3
```

```
static constexpr UInt boundary_dimension = 2
```

```
static constexpr UInt voigt = voigt_size<3>::value
```

```
static const std::vector<UInt> indices = {0}
```

```
class ModelDumper
```

```
    #include <model_dumper.hh>
```

Public Functions

```
virtual ~ModelDumper () = default
```

```
virtual void dump (const Model &model) = 0
```

```
class ModelFactory
```

```
    #include <model_factory.hh> Factory class for model.
```

Public Static Functions

static std::unique_ptr<*Model*> **createModel** (*model_type* type, const std::vector<*Real*> &system_size, const std::vector<*UInt*> &discretization)

Create new model.

static std::unique_ptr<*Model*> **createModel** (const *Model* &model)

Make a deep copy of existing model.

static std::unique_ptr<*Residual*> **createResidual** (*Model* &model, *Real* sigma_y, *Real* hardening)

Create a plasticity residual.

static void **registerVolumeOperators** (*Model* &m)

Register volume integral operators in a model.

static void **registerNonPeriodic** (*Model* &m, std::string name)

static void **setIntegrationMethod** (*IntegralOperator* &op, *integration_method* method, *Real* cutoff)

Set integration method for a volume integral operator.

template<*model_type* type>

class **ModelTemplate** : public *tamaas::Model*

#include <model_template.hh> *Model* class templated with model type Specializations of this class should take care of dimension specific code.

Public Functions

ModelTemplate (std::vector<*Real*> system_size, std::vector<*UInt*> discretization)

Constructor.

inline virtual *model_type* **getType** () const override

Get model type.

virtual std::vector<*UInt*> **getGlobalDiscretization** () const override

Get discretization of global MPI system.

virtual std::vector<*UInt*> **getBoundaryDiscretization** () const override

Get boundary discretization.

virtual std::vector<*Real*> **getBoundarySystemSize** () const override

Get boundary system physical size.

virtual void **setIntegrationMethod** (*integration_method* method, *Real* cutoff) override

Set integration method for registered volume operators.

Protected Functions

```
void initializeBEEngine ()
```

Protected Attributes

```
std::unique_ptr<ViewType> displacement_view = nullptr
```

```
std::unique_ptr<ViewType> traction_view = nullptr
```

Private Types

```
using trait = model_type_traits<type>
```

```
using ViewType = GridView<Grid, Real, dim, trait::boundary_dimension>
```

Private Static Attributes

```
static constexpr UInt dim = trait::dimension
```

```
class nan_error : public runtime_error  
    #include <errors.hh>
```

```
struct no_convergence_error : public runtime_error  
    #include <errors.hh>
```

Public Functions

```
inline no_convergence_error (const std::string &what, double error, double tolerance)
```

```
inline const char *what () const noexcept override
```

Public Members

```
double error
```

```
double tolerance
```

```
class not_implemented_error : public runtime_error  
    #include <errors.hh>  
template<UInt dim>
```

struct **Partitioner**

#include <partitioner.hh>

Public Static Functions

```
template<typename Container>
static inline decltype(auto) global_size (Container local)

template<typename T>
static inline decltype(auto) global_size (const Grid<T, dim> &grid)

template<typename Container>
static inline decltype(auto) local_size (Container global)

template<typename T>
static inline decltype(auto) local_size (const Grid<T, dim> &grid)

static inline decltype(auto) local_size (std::initializer_list<UInt> list)

template<typename Container>
static inline decltype(auto) local_offset (const Container &global)

template<typename T>
static inline decltype(auto) local_offset (const Grid<T, dim> &grid)

static inline decltype(auto) local_offset (std::initializer_list<UInt> list)

template<typename Container>
static inline decltype(auto) cast_size (const Container &s)

static inline decltype(auto) alloc_size (const std::array<UInt, dim> &global, UInt howmany)

template<typename T>
static inline Grid<T, dim> gather (const Grid<T, dim> &send)

template<typename T>
static inline Grid<T, dim> scatter (const Grid<T, dim> &send)
```

struct **plan**

#include <cuFFT_engine.hh>

Public Functions

```
plan () = default

inline plan (plan &&o) noexcept

inline plan &operator= (plan &&o) noexcept

inline ~plan () noexcept
    Destroy plan.

inline operator cuFFTHandle () const
    For seamless use with fftw api.
```

Public Members

cufftHandle **_plan**

template<typename **T**>

struct **plan**

#include <interface_impl.hh> Holder type for fftw plans.

Public Functions

inline explicit **plan** (typename *helper<T>::plan* _plan = nullptr)

Create from plan.

inline **plan** (*plan* &&o) noexcept

Move constructor to avoid accidental plan destruction.

inline *plan* &**operator=** (*plan* &&o) noexcept

Move operator.

inline ~**plan** () noexcept

Destroy plan.

inline **operator** typename *helper<T>::plan* () const

For seamless use with fftw api.

Public Members

helper<T>::plan **_plan**

class **PolonskyKeerRey** : public *tamaas::ContactSolver*

#include <polonsky_keer_rey.hh> Subclassed by *tamaas::KatoSaturated*

Public Types

enum **type**

Types of algorithm (primal/dual) or constraint.

Values:

enumerator **gap**

enumerator **pressure**

Public Functions

PolonskyKeerRey (*Model* &model, const *GridBase*<*Real*> &surface, *Real* tolerance, *type* variable_type, *type* constraint_type)

Constructor.

~PolonskyKeerRey () override = default

virtual *Real* **solve** (std::vector<*Real*> target) override

Solve.

virtual *Real* **meanOnUnsaturated** (const *GridBase*<*Real*> &field) const

Mean on unsaturated constraint zone.

Computes $\frac{1}{\text{card}(\{p>0\})} \sum_{\{p>0\}} f_i$

virtual *Real* **computeSquaredNorm** (const *GridBase*<*Real*> &field) const

Compute squared norm.

Computes $\sum_{\{p>0\}} f_i^2$

virtual void **updateSearchDirection** (*Real* factor)

Update search direction.

Do $\mathbf{t} = \mathbf{q}' + \delta \frac{R}{R_{\text{old}}} \mathbf{t}$

virtual *Real* **computeCriticalStep** (*Real* target = 0)

Compute critical step.

Computes $\tau = \frac{\sum_{\{p>0\}} q'_i t_i}{\sum_{\{p>0\}} r'_i t_i}$

virtual bool **updatePrimal** (*Real* step)

Update primal and check non-admissible state.

Update steps:

i. $\mathbf{p} = \mathbf{p} - \tau \mathbf{t}$

ii. Truncate all p negative

iii. For all points in $I_{\text{na}} = \{p = 0 \wedge q < 0\}$ do $p_i = p_i - \tau q_i$

virtual *Real* **computeError** () const

Compute error/stopping criterion.

Error is based on $\sum p_i q_i$

virtual void **enforceAdmissibleState** ()

Enforce contact constraints on final state.

virtual void **enforceMeanValue** (*Real* mean)

Enfore mean value constraint.

void **setIntegralOperator** (std::string name)

Set integral operator for gradient computation.

void **setupFunctional** ()

Set functionals for contact.

Protected Attributes

type **variable_type**

type **constraint_type**

model_type **operation_type**

GridBase<Real> ***primal** = nullptr
non-owning pointer for primal variable

GridBase<Real> ***dual** = nullptr
non-owning pointer for dual variable

std::unique_ptr<*GridBase<Real>*> **search_direction** = nullptr
CG search direction.

std::unique_ptr<*GridBase<Real>*> **projected_search_direction** = nullptr
Projected CG search direction.

std::unique_ptr<*GridBase<Real>*> **pressure_view**
View on normal pressure, gap, displacement.

std::unique_ptr<*GridBase<Real>*> **gap_view**

std::unique_ptr<*GridBase<Real>*> **displacement_view** = nullptr

std::shared_ptr<*IntegralOperator*> **integral_op** = nullptr
Integral operator for gradient computation.

Private Functions

template<*model_type* **type**>
void **setViews** ()
Set correct views on normal traction and gap.

template<*model_type* **type**>
void **defaultOperator** ()
Set the default integral operator.

class **PolonskyKeerTan** : public *tamaas::Kato*
#include <polonsky_keer_tan.hh>

Public Functions

PolonskyKeerTan (*Model* &model, const *GridBase*<*Real*> &surface, *Real* tolerance, *Real* mu)

Constructor.

Real **solve** (*GridBase*<*Real*> &p0)

Solve with Coulomb friction.

Real **solveTresca** (*GridBase*<*Real*> &p0)

Solve with Tresca friction.

template<*model_type* **type**>

Real **solveTpl** (*GridBase*<*Real*> &p0, bool use_tresca = false)

Template for solve function.

template<*UInt* **comp**>

void **enforcePressureMean** (*GridBase*<*Real*> &p0)

Enforce pressure mean.

template<*UInt* **comp**>

Vector<*Real*, *comp*> **computeMean** (*GridBase*<*Real*> &field, bool on_c)

Compute mean of field (only on I_c)

Real **computeSquaredNorm** (*GridBase*<*Real*> &field)

Compute squared norm.

template<*UInt* **comp**>

void **truncateSearchDirection** (bool on_c)

Restrict search direction on I_c.

template<*UInt* **comp**>

Real **computeStepSize** (bool on_c)

Compute optimal step size (only on I_c)

Private Members

std::unique_ptr<*GridBase*<*Real*>> **search_direction** = nullptr

std::unique_ptr<*GridBase*<*Real*>> **search_direction_backup** = nullptr

std::unique_ptr<*GridBase*<*Real*>> **projected_search_direction** = nullptr

template<*UInt*... **ns**>

struct **product** : public *detail::product_tail_rec*<1, *ns*...>

#include <static_types.hh>

template<*UInt* **acc**, *UInt* **n**>

struct **product_tail_rec**<*acc*, *n*> : public std::integral_constant<*UInt*, *acc* * *n*>

#include <static_types.hh>

template<typename **T**>

struct **ptr**

#include <interface_impl.hh> RAII helper for `fftw_free`.

Public Functions

inline **~ptr** () noexcept

inline **operator T*** ()

Public Members

T * **_ptr**

template<typename **LocalType**, typename **ValueType**, *UInt* **local_size**>

class **Range**

#include <ranges.hh> *Range* class for complex iterators.

Public Types

using **value_type** = *LocalType*

using **reference** = *value_type*&&

Public Functions

template<class **Container**>

inline **Range** (*Container* &&cont)

Construct from a container.

inline **Range** (iterator **_begin**, iterator **_end**)

Construct from two iterators.

inline iterator **begin** ()

inline iterator **end** ()

inline *Range* **headless** () const

Private Members

iterator **_begin**

iterator **_end**

template<*operation* **op**, typename **ReturnType**>

struct **reduction_helper**

#include <loop_utils.hh>

template<typename **ReturnType**>

struct **reduction_helper**<*operation::max*, *ReturnType*> : public thrust::maximum<*ReturnType*>

#include <loop_utils.hh>

Public Functions

inline *ReturnType* **init** () const

template<typename **ReturnType**>

struct **reduction_helper**<*operation::min*, *ReturnType*> : public thrust::minimum<*ReturnType*>

#include <loop_utils.hh>

Public Functions

inline *ReturnType* **init** () const

template<typename **ReturnType**>

struct **reduction_helper**<*operation::plus*, *ReturnType*> : public thrust::plus<*ReturnType*>

#include <loop_utils.hh>

Public Functions

inline *ReturnType* **init** () const

template<typename **ReturnType**>

struct **reduction_helper**<*operation::times*, *ReturnType*> : public thrust::multiplies<*ReturnType*>

#include <loop_utils.hh>

Public Functions

inline *ReturnType* **init** () const

template<*UInt* **dim**>

class **RegularizedPowerlaw** : public *tamaas::Filter*<*dim*>

#include <regularized_powerlaw.hh> Class representing an isotropic power law spectrum.

Public Functions

virtual void **computeFilter** (*GridHermitian*<*Real*, *dim*> &filter_coefficients) const override

Compute filter coefficients.

inline *Real* **operator** () (const *VectorProxy*<*Real*, *dim*> &q_vec) const

Compute a point of the PSD.

TAMAAS_ACCESSOR (*q1*, *UInt*, Q1)

TAMAAS_ACCESSOR (*q2*, *UInt*, Q2)

TAMAAS_ACCESSOR (*hurst*, *Real*, Hurst)

Protected Attributes

UInt **q1**

UInt **q2**

Real **hurst**

class **Residual**

#include <residual.hh> *Residual* manager.

Public Functions

Residual (*Model* &model, std::shared_ptr<*Material*> material)

Constructor.

virtual **~Residual** () = default

Destructor.

virtual void **computeResidual** (*GridBase*<*Real*> &strain_increment)

Compute the residual vector for a given strain increment.

virtual void **computeResidualDisplacement** (*GridBase*<*Real*> &strain_increment)

Compute residual surface displacement.

virtual void **applyTangent** (*GridBase*<*Real*> &output, *GridBase*<*Real*> &input, *GridBase*<*Real*> &strain_increment)

Apply tangent to arbitrary increment.

virtual void **updateState** (*GridBase<Real>* &converged_strain_increment)

Update the plastic state.

inline const *Model* &**getModel** () const

Return model reference.

inline *Model* &**getModel** ()

Return model reference (non-const)

inline const *GridBase<Real>* &**getVector** () const

Return residual vector.

inline const *Material* &**getMaterial** () const

Return material.

Protected Attributes

Model &**model**

std::shared_ptr<*Material*> **material**

std::shared_ptr<*Grid<Real, 3>>* **strain**

std::shared_ptr<*Grid<Real, 3>>* **stress**

std::shared_ptr<*Grid<Real, 3>>* **residual**

std::shared_ptr<*Grid<Real, 3>>* **tmp**

std::unordered_set<*UInt*> **plastic_layers**

std::function<bool(*UInt*)> **plastic_filter**

Private Functions

inline const *IntegralOperator* &**integralOperator** (const std::string &name) const

Convenience function.

void **updateFilter** (*Grid<Real, dim>* &plastic_strain_increment)

Add non-zero layers of plastic strain into the filter.

Private Static Attributes

```
static constexpr auto type = model_type::volume_2d
```

```
static constexpr auto dim = model_type_traits<type>::dimension
```

```
static constexpr auto voigt = model_type_traits<type>::voigt
```

```
struct sequential  
    #include <mpi_interface.hh>
```

Public Static Functions

```
static inline void enter ()
```

```
static inline void exit ()
```

```
struct sequential_guard  
    #include <mpi_interface.hh>
```

Public Functions

```
inline sequential_guard ()
```

```
inline ~sequential_guard ()
```

```
template<typename T>
```

```
struct span  
    #include <span.hh>
```

Public Types

```
using element_type = T
```

```
using value_type = std::remove_cv_t<T>
```

```
using size_type = std::size_t
```

```
using difference_type = std::ptrdiff_t
```

```
using pointer = T*
```

```
using const_pointer = const T*
```

```
using reference = T&

using const_reference = const T&

using iterator = T*

using reverse_iterator = std::reverse_iterator<iterator>
```

Public Functions

```
inline constexpr size_type size () const noexcept

inline constexpr pointer data () const noexcept

inline constexpr iterator begin () const noexcept

inline constexpr iterator end () const noexcept

inline constexpr iterator begin () noexcept

inline constexpr iterator end () noexcept

inline reference operator [] (size_type idx)

inline const_reference operator [] (size_type idx) const
```

Public Members

```
pointer data_ = nullptr
```

```
size_type size_ = 0
```

```
class SquaredExponentialAdhesionFunctional : public tamaas::functional::AdhesionFunctional
    #include <adhesion_functional.hh> Squared exponential adhesion functional.
```

Public Functions

```
inline SquaredExponentialAdhesionFunctional (const GridBase<Real> &surface)
    Explicit declaration of constructor for swig.

virtual Real computeF (GridBase<Real> &gap, GridBase<Real> &pressure) const override
    Compute the total adhesion energy.

virtual void computeGradF (GridBase<Real> &gap, GridBase<Real> &gradient) const override
    Compute the gradient of the adhesion functional.

template<typename, typename, UInt...> class StaticParent, UInt... dims
```

```

struct static_size_helper : public tamaas::product<dims...>
    #include <static_types.hh>

template<UInt n>

struct static_size_helper<StaticSymMatrix, n> : public tamaas::voigt_size<n>
    #include <static_types.hh>

template<typename DataType, typename SupportType, UInt _size>

class StaticArray

```

#include <static_types.hh> *Static Array*.

This class is meant to be a small and fast object for intermediate calculations, possibly on wrapped memory belonging to a grid. Support type show be either a pointer or a C array. It should not contain any virtual method.

Public Types

using **value_type** = *T*

Public Functions

StaticArray () = default

~StaticArray () = default

StaticArray (const *StaticArray*&) = delete

StaticArray (*StaticArray*&&) = delete

StaticArray &**operator**= (*StaticArray*&&) = delete

inline auto **operator** () (U**Int** i) -> *T*&

Access operator.

inline auto **operator** () (U**Int** i) const -> const *T*&

Access operator.

template<typename **DT**, typename **ST**>

inline auto **dot** (const *StaticArray*<*DT*, *ST*, *size*> &o) const -> *T_bare*

Scalar product.

inline *T_bare* **l2squared** () const

L2 norm squared.

inline *T_bare* **l2norm** () const

L2 norm.

inline *T_bare* **sum** () const

Sum of all elements.

template<typename **DT**,

typename **ST**> inline __host__ void **operator**+= (const *StaticArray*< *DT*, *ST*,
size > &o)

```
template<typename DT,
typename ST> inline __host__ void operator-= (const StaticArray< DT, ST,
size > &o)
```

```
template<typename DT,
typename ST> inline __host__ void operator*= (const StaticArray< DT, ST,
size > &o)
```

```
template<typename DT, typename ST> inline __host__ void operator/
= (const StaticArray< DT, ST, size > &o)
```

```
template<typename T1> inline __host__ std::enable_if_t< is_arith-
metic< T1 >::value, StaticArray & > operator+= (const T1 &x)
```

```
template<typename T1> inline __host__ std::enable_if_t< is_arith-
metic< T1 >::value, StaticArray & > operator-= (const T1 &x)
```

```
template<typename T1> inline __host__ std::enable_if_t< is_arith-
metic< T1 >::value, StaticArray & > operator*= (const T1 &x)
```

```
template<typename T1> inline __host__ std::enable_if_t< is_arith-
metic< T1 >::value, StaticArray & > operator/= (const T1 &x)
```

```
template<typename T1> inline __host__ std::enable_if_t< is_arith-
metic< T1 >::value, StaticArray & > operator= (const T1 &x)
```

```
inline StaticArray &operator= (const StaticArray &o)
```

Overriding the implicit copy operator.

```
template<typename DT, typename ST>
inline void operator= (const StaticArray<DT, ST, size> &o)
```

```
template<typename DT, typename ST>
inline StaticArray &copy (const StaticArray<DT, ST, size> &o)
```

```
inline T *begin ()
```

```
inline const T *begin () const
```

```
inline T *end ()
```

```
inline const T *end () const
```

```
inline valid_size_t<T&> front ()
```

```
inline valid_size_t<const T&> front () const
```

```
inline valid_size_t<T&> back ()
```

```
inline valid_size_t<const T&> back () const
```

Public Static Attributes

static constexpr *UInt* **size** = *_size*

Protected Attributes

SupportType **_mem**

Private Types

using **T** = *DataType*

using **T_bare** = typename std::remove_cv_t<*T*>

template<typename **U**>

using **valid_size_t** = std::enable_if_t<(*size* > 0), *U*>

template<typename **DataType**, typename **SupportType**, *UInt* **n**, *UInt* **m**>

class **StaticMatrix** : public *tamaas::StaticTensor*<*DataType*, *SupportType*, *n*, *m*>

#include <static_types.hh>

Public Functions

template<typename **DT**, typename **ST**>

std::enable_if_t<*n* == *m*> **fromSymmetric** (const *StaticSymMatrix*<*DT*, *ST*, *n*> &o)

template<typename **DT1**, typename **ST1**, typename **DT2**, typename **ST2**>

void **outer** (const *StaticVector*<*DT1*, *ST1*, *n*> &a, const *StaticVector*<*DT2*, *ST2*, *m*> &b)

Outer product of two vectors.

template<typename **DT1**, typename **ST1**, typename **DT2**, typename **ST2**, *UInt* **l**>

inline void **mul** (const *StaticMatrix*<*DT1*, *ST1*, *n*, *l*> &a, const *StaticMatrix*<*DT2*, *ST2*, *l*, *m*> &b)

inline std::enable_if_t<*n* == *m*, *T_bare*> **trace** () const

template<typename **DT1**, typename **ST1**>

inline std::enable_if_t<*n* == *m*> **deviatoric** (const *StaticMatrix*<*DT1*, *ST1*, *n*, *m*> &mat, *Real* factor = *n*)

Private Types

using **T** = *DataType*

using **T_bare** = typename std::remove_cv_t<*T*>

template<typename **DataType**, typename **SupportType**, *UInt* **n**>

class **StaticSymMatrix**

#include <static_types.hh> Symmetric matrix in Voigt notation.

Public Functions

template<typename **DT**, typename **ST**>

inline void **symmetrize** (const *StaticMatrix*<*DT*, *ST*, *n*, *n*> &m)

Copy values from matrix and symmetrize.

template<typename **DT**, typename **ST**>

inline void **operator+=** (const *StaticMatrix*<*DT*, *ST*, *n*, *n*> &m)

Add values from symmetrized matrix.

inline auto **trace** () const

template<typename **DT**, typename **ST**>

inline void **deviatoric** (const *StaticSymMatrix*<*DT*, *ST*, *n*> &m, *Real* factor = *n*)

Private Types

using **parent** = *StaticVector*<*DataType*, *SupportType*, *voigt_size*<*n*>::value>

using **T** = std::remove_cv_t<*DataType*>

Private Functions

template<typename **DT**, typename **ST**, typename **BinOp**>

inline void **sym_binary** (const *StaticMatrix*<*DT*, *ST*, *n*, *n*> &m, *BinOp* &&op)

template<typename **DataType**, typename **SupportType** = *DataType**, *UInt*... **dims**>

class **StaticTensor** : public *tamaas::StaticArray*<*DataType*, *DataType**, *product*<*dims*...>::value>

#include <static_types.hh> Static *Tensor*.

This class implements a multi-dimensional tensor behavior.

Public Functions

```
template<typename ...Idx>
inline const T &operator () (Idx... idx) const
```

```
template<typename ...Idx>
inline T &operator () (Idx... idx)
```

Public Static Attributes

```
static constexpr UInt dim = sizeof...(dims)
```

Private Types

```
using parent = StaticArray<DataType, SupportType, product<dims...>::value>
```

```
using T = DataType
```

Private Static Functions

```
template<typename ...Idx>
static inline UInt unpackOffset (UInt offset, UInt index, Idx... rest)
```

```
template<typename ...Idx>
static inline UInt unpackOffset (UInt offset, UInt index)
```

```
template<typename DataType, typename SupportType, UInt n>
```

```
class StaticVector
```

```
#include <static_types.hh> Vector class with size determined at compile-time.
```

Public Functions

```
template<bool transpose, typename DT1, typename ST1, typename DT2, typename ST2, UInt m>
inline void mul (const StaticMatrix<DT1, ST1, n, m> &mat, const StaticVector<DT2, ST2, m> &vec)
```

Matrix-vector product.

Private Types

```
using T = std::remove_cv_t<DataType>
```

```
template<UInt dim>
```

```
struct Statistics
```

```
#include <statistics.hh> Suitcase class for all statistics related functions.
```

Public Types

using **PVector** = *VectorProxy*<*Real*, *dim*>

Public Functions

std::vector<*Real*> **computeMoments** (*Grid*<*Real*, 1> &surface)

std::vector<*Real*> **computeMoments** (*Grid*<*Real*, 2> &surface)

Public Static Functions

static *Real* **computeRMSHeights** (*Grid*<*Real*, *dim*> &surface)

Compute hrms.

static *Real* **computeSpectralRMSSlope** (*Grid*<*Real*, *dim*> &surface)

Compute hrms' in fourier space.

static *Real* **computeFDRMSSlope** (*Grid*<*Real*, *dim*> &surface)

Compute hrms' with finite differences.

static *GridHermitian*<*Real*, *dim*> **computePowerSpectrum** (*Grid*<*Real*, *dim*> &surface)

Compute PSD of surface.

static *Grid*<*Real*, *dim*> **computeAutocorrelation** (*Grid*<*Real*, *dim*> &surface)

Compute autocorrelation.

static std::vector<*Real*> **computeMoments** (*Grid*<*Real*, *dim*> &surface)

Compute spectral moments.

static *Real* **contact** (const *Grid*<*Real*, *dim*> &tractions, *UInt* perimeter = 0)

Compute (corrected) contact area fraction.

static *Real* **graphArea** (const *Grid*<*Real*, *dim*> &displacement)

Compute the area scaling factor of a periodic graph.

Private Static Functions

template<class **T**>

static *Real* **rmsSlopesFromPSD** (const *GridHermitian*<*Real*, *dim*> &psd, const *Grid*<*T*, *dim*> &diff)

template<*UInt dim*>

class **SurfaceGenerator**

#include <surface_generator.hh> Class generating random surfaces.

Subclassed by *tamaas::SurfaceGeneratorFilter*< *dim* >

Public Functions

SurfaceGenerator () = default
Default constructor.

inline **SurfaceGenerator** (std::array<UInt, dim> global_size)
Construct with surface global size.

virtual ~**SurfaceGenerator** () = default
Default destructor.

virtual *Grid*<Real, dim> &**buildSurface** () = 0
Build surface profile (array of heights)

void **setSize** (std::array<UInt, dim> n)
Set surface sizes.

void **setSize** (const UInt n[dim])
Set surface sizes.

inline auto **getSize** () const
Get surface sizes.

inline long **getRandomSeed** () const

void **setRandomSeed** (long seed)

Protected Attributes

Grid<Real, dim> **grid**

std::array<UInt, dim> **global_size** = {0}

long **random_seed** = 0

template<UInt dim>

class **SurfaceGeneratorFilter** : public *tamaas::SurfaceGenerator*<dim>
#include <surface_generator_filter.hh> Subclassed by *tamaas::SurfaceGeneratorRandomPhase*< dim >

Public Functions

virtual *Grid*<Real, dim> &**buildSurface** () override
Construct with surface size.
Build surface with Hu & Tonder algorithm

inline void **setFilter** (std::shared_ptr<Filter<dim>> new_filter)
Set filter object.

inline void **setSpectrum** (std::shared_ptr<Filter<dim>> spectrum)
Set spectrum.

inline const *Filter*<dim> ***getSpectrum** () const
Get spectrum.

Protected Functions

```
void applyFilterOnSource ()  
    Apply filter coefficients on white noise.  
  
template<typename T>  
void generateWhiteNoise ()  
    Generate white noise with given distribution.
```

Protected Attributes

```
std::shared_ptr<Filter<dim>> filter = nullptr  
  
GridHermitian<Real, dim> filter_coefficients  
  
Grid<Real, dim> white_noise  
  
std::unique_ptr<FFTEngine> engine = FFTEngine::makeEngine()  
  
template<UInt dim>  
class SurfaceGeneratorRandomPhase : public tamaas::SurfaceGeneratorFilter<dim>  
    #include <surface_generator_random_phase.hh>
```

Public Functions

```
virtual Grid<Real, dim> &buildSurface () override  
    Build surface with uniform random phase.  
  
template<model_type type>  
struct SurfaceTractionHelper  
    #include <kelvin_helper.hh>
```

Public Types

```
using trait = model_type_traits<type>  
  
using BufferType = GridHermitian<Real, bdim>  
  
using kelvin_t = influence::Kelvin<3, 2>  
  
using source_t = typename KelvinTrait<kelvin_t>::source_t  
  
using out_t = typename KelvinTrait<kelvin_t>::out_t
```

Public Functions

```
template<bool apply_q_power>
inline void computeSurfaceTractions (std::vector<BufferType> &source, BufferType &tractions, const
                                     Grid<Real, bdim> &wavevectors, Real domain_size, const
                                     kelvin_t &kelvin, const influence::ElasticHelper<dim> &el)
```

Compute surface tractions due to eigenstress distribution.

Public Static Attributes

```
static constexpr UInt dim = trait::dimension
```

```
static constexpr UInt bdim = trait::boundary_dimension
```

Protected Attributes

```
Accumulator<type, source_t> accumulator
```

```
template<UInt dim>
```

```
struct SymHookeFieldHelper
```

Public Functions

```
inline void operator () (SymMatrixProxy<Real, dim> sigma, SymMatrixProxy<const Real, dim> epsilon, const
                        Real &mu, const Real &nu)
```

Public Members

```
influence::ElasticHelper<dim> elasticity = {0, 0}
```

```
struct TamaasInfo
```

```
#include <tamaas_info.hh>
```

Public Static Attributes

```
static const std::string version
```

```
static const std::string build_type
```

```
static const std::string branch
```

```
static const std::string commit
```

static const std::string **remotes**

static const std::string **diff**

static const std::string **backend**

static const bool **has_mpi**

template<template<typename, typename, *UInt*...> class **StaticParent**, typename **T**, *UInt*... **dims**>

class **Tensor** : public *StaticParent*<**T**, *T*[*static_size_helper*<*StaticParent*, *dims*...>::value], *dims*...>

#include <static_types.hh>

Public Functions

Tensor () = default

Default constructor.

inline **Tensor** (*T* val)

Construct with default value.

inline **Tensor** (const std::array<*T*, *size*> &arr)

Construct from array.

inline *Tensor* &**operator=** (const std::array<*T*, *size*> &arr)

Copy from array.

template<typename **DT**, typename **ST**>

inline **Tensor** (const *StaticParent*<*DT*, *ST*, *dims*...> &o)

Construct by copy from static tensor.

inline **Tensor** (const *Tensor* &o)

inline *Tensor* &**operator=** (const *Tensor* &o)

inline **Tensor** (*Tensor* &&o) noexcept

Private Types

using **parent** = *StaticParent*<*T*, *T*[*size*], *dims*...>

Private Static Attributes

static constexpr *UInt* **size** = *static_size_helper*<*StaticParent*, *dims*...>::value

template<template<typename, typename, *UInt*...> class **StaticParent**, typename **T**, *UInt*... **dims**>

class **TensorProxy** : public *StaticParent*<*T*, *T**, *dims*...>

#include <static_types.hh> Proxy type for tensor.

Public Types

using **stack_type** = *Tensor*<*StaticParent*, *T*, *dims...*>

Public Functions

inline explicit **TensorProxy** (*T* *spot)

Explicit construction from data location.

inline explicit **TensorProxy** (*T* &spot)

Explicit construction from lvalue-reference.

template<typename **DataType**, typename **SupportType**>

inline **TensorProxy** (*StaticParent*<*DataType*, *SupportType*, *dims...*> &o)

Construction from static tensor.

inline **TensorProxy** (const *TensorProxy* &o)

inline *TensorProxy* &**operator=** (const *TensorProxy* &o)

inline **TensorProxy** (*TensorProxy* &&o) noexcept

Private Types

using **parent** = *StaticParent*<*T*, *T**, *dims...*>

struct **ToleranceManager**

#include <ep_solver.hh>

Public Functions

inline **ToleranceManager** (*Real* start, *Real* end, *Real* rate)

inline void **step** ()

inline void **reset** ()

Public Members

Real **start_tol**

Real **end_tol**

Real **rate**

Real **tolerance**

template<typename **Iterator**, typename **Functor**, typename **Value**>

```
class transform_iterator : public thrust::iterator_adaptor<transform_iterator<Iterator, Functor, Value>,
Iterator, Value, thrust::use_default, thrust::use_default, Value>
```

#include <loop.hh> Replacement for thrust::transform_iterator which copies values.

Public Types

```
using super_t = thrust::iterator_adaptor<transform_iterator<Iterator, Functor, Value>, Iterator, Value,
thrust::use_default, thrust::use_default, Value>
```

Public Functions

```
inline transform_iterator (const Iterator &x, const Functor &func)
```

Private Functions

```
inline auto dereference () const -> decltype(func(*this->base()))
```

Private Members

Functor **func**

Friends

```
friend class thrust::iterator_core_access
```

```
template<typename T>
```

```
struct UnifiedAllocator
```

#include <unified_allocator.hh> Class allocating unified memory

Public Static Functions

```
static inline span<T> allocate (typename span<T>::size_type n) noexcept
```

Allocate memory.

```
static inline void deallocate (span<T> view) noexcept
```

Free memory.

```
template<U dim>
```

```
struct voigt_size
```

#include <static_types.hh>

```
template<>
```

```
struct voigt_size<1> : public std::integral_constant<UInt, 1>
    #include <static_types.hh>
template<>
struct voigt_size<2> : public std::integral_constant<UInt, 3>
    #include <static_types.hh>
template<>
struct voigt_size<3> : public std::integral_constant<UInt, 6>
    #include <static_types.hh>
template<model_type type>
class VolumePotential : public tamaas::IntegralOperator
    #include <volume_potential.hh> Volume potential operator class. Applies the operators for computation of dis-
    placements and strains due to residual/eigen strains.
    Subclassed by tamaas::Boussinesq< type, derivative >, tamaas::Kelvin< type, derivative >
```

Public Functions

```
VolumePotential (Model *model)
inline virtual void updateFromModel () override
    Update from model (does nothing)
inline virtual IntegralOperator::kind getKind () const override
    Kind.
virtual model_type getType () const override
    Type.
inline virtual void apply (GridBase<Real> &input, GridBase<Real> &output) const override
    Apply to all of the source layers.
```

Protected Types

```
using filter_t = const std::function<bool(UInt)>&
using BufferType = GridHermitian<Real, trait::boundary_dimension>
```

Protected Functions

void **transformSource** (*GridBase<Real>* &in, *filter_t* pred) const

Transform source layer-by-layer.

inline void **transformSource** (*GridBase<Real>* &in) const

Transform all source.

template<typename **Func**>

void **transformOutput** (*Func* func, *GridBase<Real>* &out) const

Transform output layer-by-layer.

void **initialize** (*UInt* source_components, *UInt* out_components, *UInt* out_buffer_size)

Initialize fourier buffers.

Protected Attributes

Grid<Real, trait::boundary_dimension> **wavevectors**

mutable std::vector<*BufferType*> **source_buffer**

mutable std::vector<*BufferType*> **out_buffer**

mutable std::unique_ptr<*FFTEngine*> **engine**

Private Types

using **trait** = *model_type_traits<type>*

struct **VonMises**

#include <computes.hh> Compute von Mises stress on a tensor field.

Public Static Functions

template<*UInt* **dim**>

static inline void **call** (*Grid<Real, dim>* &vm, const *Grid<Real, dim>* &stress)

template<*model_type* **mtype**, *IntegralOperator::kind* **otype**>

class **Westergaard** : public *tamaas::IntegralOperator*

#include <westergaard.hh> Operator based on *Westergaard* solution and the Discrete Fourier Transform. This class is templated with model type to allow efficient storage of the influence coefficients. The integral operator is only applied to surface pressure/displacements, even for volume models.

Public Functions

Westergaard (*Model* *model)

Constructor: initializes influence coefficients and allocates buffer.

inline const *GridHermitian*<*Real*, *bdim*> &**getInfluence** () const

Get influence coefficients.

virtual void **apply** (*GridBase*<*Real*> &input, *GridBase*<*Real*> &output) const override

Apply influence coefficients to input.

inline virtual void **updateFromModel** () override

Update the influence coefficients.

inline virtual *IntegralOperator::kind* **getKind** () const override

Kind.

inline virtual *model_type* **getType** () const override

Type.

void **initInfluence** ()

Initialize influence coefficients.

template<typename **Functor**>

void **initFromFunctor** (*Functor* func)

template<typename **Functor**>

void **fourierApply** (*Functor* func, *GridBase*<*Real*> &in, *GridBase*<*Real*> &out) const

Apply a functor in Fourier space.

virtual *Real* **getOperatorNorm** () override

Compute L₂ norm of influence functions.

virtual std::pair<*UInt*, *UInt*> **matvecShape** () const override

Dense shape.

virtual *GridBase*<*Real*> **matvec** (*GridBase*<*Real*> &x) const override

Dense matvec.

Public Members

std::shared_ptr<*GridHermitian*<*Real*, *bdim*>> **influence**

mutable *GridHermitian*<*Real*, *bdim*> **buffer**

mutable std::unique_ptr<*FFTEngine*> **engine**

Private Types

```
using trait = model_type_traits<mtype>
```

Private Static Attributes

```
static constexpr UInt dim = trait::dimension
```

```
static constexpr UInt bdim = trait::boundary_dimension
```

```
static constexpr UInt comp = trait::components
```

```
namespace cufft
```

```
namespace fftw
```

```
namespace fftw_impl
```

Functions

```
template<typename T>  
inline auto free (T *ptr)
```

Free memory.

```
inline auto init_threads ()
```

Init FFTW with threads.

```
inline auto plan_with_nthreads (int nthreads)
```

Set number of threads.

```
inline auto cleanup_threads ()
```

Cleanup threads.

```
inline auto plan_many_forward (int rank, const int *n, int howmany, double *in, const int *inembed, int  
istride, int idist, fftw_complex *out, const int *onembed, int ostride, int  
odist, unsigned flags)
```

```
inline auto plan_many_backward (int rank, const int *n, int howmany, fftw_complex *in, const int  
*inembed, int istride, int idist, double *out, const int *onembed, int  
ostride, int odist, unsigned flags)
```

```
inline auto plan_1d_forward (int n, double *in, fftw_complex *out, unsigned flags)
```

```
inline auto plan_1d_backward (int n, fftw_complex *in, double *out, unsigned flags)
```

```
inline auto plan_2d_forward (int n0, int n1, double *in, fftw_complex *out, unsigned flags)
```

```
inline auto plan_2d_backward (int n0, int n1, fftw_complex *out, double *in, unsigned flags)
```

```

inline auto execute (fftw_plan plan)

inline auto execute (fftw_plan plan, double *in, fftw_complex *out)

inline auto execute (fftw_plan plan, fftw_complex *in, double *out)

inline auto destroy (fftw_plan plan)

inline auto plan_many_forward (int rank, const int *n, int howmany, long double *in, const int *inembed,
                                int istride, int idist, fftwl_complex *out, const int *onembed, int ostride,
                                int odist, unsigned flags)

inline auto plan_many_backward (int rank, const int *n, int howmany, fftwl_complex *in, const int
                                *inembed, int istride, int idist, long double *out, const int *onembed, int
                                ostride, int odist, unsigned flags)

inline auto plan_1d_forward (int n, long double *in, fftwl_complex *out, unsigned flags)

inline auto plan_1d_backward (int n, fftwl_complex *in, long double *out, unsigned flags)

inline auto plan_2d_forward (int n0, int n1, long double *in, fftwl_complex *out, unsigned flags)

inline auto plan_2d_backward (int n0, int n1, fftwl_complex *out, long double *in, unsigned flags)

inline auto execute (fftwl_plan plan)

inline auto execute (fftwl_plan plan, long double *in, fftwl_complex *out)

inline auto execute (fftwl_plan plan, fftwl_complex *in, long double *out)

inline auto destroy (fftwl_plan plan)

inline auto plan_many_forward (int rank, const int *n, int howmany, float *in, const int *inembed, int
                                istride, int idist, fftwf_complex *out, const int *onembed, int ostride, int
                                odist, unsigned flags)

inline auto plan_many_backward (int rank, const int *n, int howmany, fftwf_complex *in, const int
                                *inembed, int istride, int idist, float *out, const int *onembed, int ostride,
                                int odist, unsigned flags)

inline auto plan_1d_forward (int n, float *in, fftwf_complex *out, unsigned flags)

inline auto plan_1d_backward (int n, fftwf_complex *in, float *out, unsigned flags)

inline auto plan_2d_forward (int n0, int n1, float *in, fftwf_complex *out, unsigned flags)

inline auto plan_2d_backward (int n0, int n1, fftwf_complex *out, float *in, unsigned flags)

inline auto execute (fftwf_plan plan)

inline auto execute (fftwf_plan plan, float *in, fftwf_complex *out)

inline auto execute (fftwf_plan plan, fftwf_complex *in, float *out)

inline auto destroy (fftwf_plan plan)

```

```
namespace mpi_dummy
```

Functions

```
inline void init ()
```

```
inline void cleanup ()
```

```
inline auto local_size_many (int rank, const std::ptrdiff_t *size, std::ptrdiff_t howmany)
```

```
namespace tamaas
```

Typedefs

```
template<typename T>
```

```
using Allocator = FFTWAllocator<T>
```

```
template<typename T, UInt n, UInt m>
```

```
using Matrix = Tensor<StaticMatrix, T, n, m>
```

```
template<typename T, UInt n>
```

```
using SymMatrix = Tensor<StaticSymMatrix, T, n>
```

```
template<typename T, UInt n>
```

```
using Vector = Tensor<StaticVector, T, n>
```

```
template<typename T, UInt n, UInt m>
```

```
using MatrixProxy = TensorProxy<StaticMatrix, T, n, m>
```

```
template<typename T, UInt n>
```

```
using SymMatrixProxy = TensorProxy<StaticSymMatrix, T, n>
```

```
template<typename T, UInt n>
```

```
using VectorProxy = TensorProxy<StaticVector, T, n>
```

```
using Real = double
```

```
    default floating point type
```

```
using Int = int
```

```
    default signed integer type
```

```
using UInt = std::make_unsigned_t<Int>
```

```
    default unsigned integer type
```

```
template<typename T>
```

```
using complex = thrust::complex<T>
```

```
    template complex wrapper
```

```
using Complex = complex<Real>  
    default floating point complex type  
  
using random_engine = ::thrust::random::default_random_engine
```

Enums

enum class **LogLevel**

Log levels enumeration.

Values:

enumerator **debug**

enumerator **info**

enumerator **warning**

enumerator **error**

enum class **operation**

Enumeration of reduction operations.

Values:

enumerator **plus**

enumerator **times**

enumerator **min**

enumerator **max**

enum class **integration_method**

Integration method enumeration.

Values:

enumerator **cutoff**

enumerator **linear**

enum class **model_type**

Types for grid dimensions and number of components.

Values:

enumerator **basic_1d**

one component line

enumerator **basic_2d**

one component surface

enumerator **surface_1d**

two components line

enumerator **surface_2d**

three components surface

enumerator **volume_1d**

two components volume

enumerator **volume_2d**

three components volume

Functions

void **eigenvalues** (*model_type* type, *GridBase<Real>* &eigs, const *GridBase<Real>* &field)

void **vonMises** (*model_type* type, *GridBase<Real>* &eigs, const *GridBase<Real>* &field)

void **deviatoric** (*model_type* type, *GridBase<Real>* &dev, const *GridBase<Real>* &field)

template<typename **Compute**>

void **applyCompute** (*model_type* type, *GridBase<Real>* &result, const *GridBase<Real>* &field)

template<typename **T**, *UInt* **dim**>

inline std::ostream &**operator<<** (std::ostream &stream, const *Grid<T, dim>* &_this)

template<template<typename, *UInt*> class **Base**, typename **T**, *UInt* **base_dim**, typename ...**Args**>
GridView<Base, T, base_dim, base_dim - sizeof...(Args)> **make_view** (*Base<T, base_dim>* &base, *Args...*
indices)

template<template<typename, *UInt*> class **Base**, typename **T**, *UInt* **base_dim**>
GridView<Base, T, base_dim, base_dim> **make_component_view** (*Base<T, base_dim>* &base, *UInt*
component)

std::ostream &**operator<<** (std::ostream &o, const *LogLevel* &val)

template<typename ...**Ranges**>

void **checkLoopSize** (*Ranges&&*... ranges)

template<typename **LocalType**, class **Container**>

std::enable_if_t<*is_proxy<LocalType>*::value, Range<*LocalType*, typename *LocalType*::value_type, *LocalType*::size>> **range** (*Container*
&&

Make range with proxy type.

template<typename **LocalType**, class **Container**>

```
std::enable_if_t<not is_proxy<LocalType>::value, Range<LocalType, LocalType, 1>> range (Container
&&cont)
```

Make range with standard type.

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt dim>
Vector<decltype(DT1(0) + DT2(0)), dim> operator+ (const StaticVector<DT1, ST1, dim> &a, const
StaticVector<DT2, ST2, dim> &b)
```

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt dim>
Vector<decltype(DT1(0) - DT2(0)), dim> operator- (const StaticVector<DT1, ST1, dim> &a, const
StaticVector<DT2, ST2, dim> &b)
```

```
template<typename DT1, typename ST1, UInt dim>
Vector<decltype(DT1(0)), dim> operator- (const StaticVector<DT1, ST1, dim> &a)
```

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n, UInt m>
Matrix<decltype(DT1(0) + DT2(0)), n, m> operator+ (const StaticMatrix<DT1, ST1, n, m> &a, const
StaticMatrix<DT2, ST2, n, m> &b)
```

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n, UInt m>
Matrix<decltype(DT1(0) - DT2(0)), n, m> operator- (const StaticMatrix<DT1, ST1, n, m> &a, const
StaticMatrix<DT2, ST2, n, m> &b)
```

```
template<typename DT1, typename ST1, UInt n, UInt m>
Matrix<decltype(DT1(0)), n, m> operator- (const StaticMatrix<DT1, ST1, n, m> &a)
```

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n>
SymMatrix<decltype(DT1(0) + DT2(0)), n> operator+ (const StaticSymMatrix<DT1, ST1, n> &a, const
StaticSymMatrix<DT2, ST2, n> &b)
```

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n>
SymMatrix<decltype(DT1(0) - DT2(0)), n> operator- (const StaticSymMatrix<DT1, ST1, n> &a, const
StaticSymMatrix<DT2, ST2, n> &b)
```

```
template<typename DT1, typename ST1, UInt dim>
SymMatrix<decltype(DT1(0)), dim> operator- (const StaticSymMatrix<DT1, ST1, dim> &a)
```

```
template<typename DT, typename ST, typename T, UInt n, typename =
std::enable_if_t<is_arithmetic<T>::value>>
Vector<decltype(DT(0) * T(0)), n> operator* (const StaticVector<DT, ST, n> &a, const T &b)
```

```
template<typename DT, typename ST, typename T, UInt n, typename =
std::enable_if_t<is_arithmetic<T>::value>>
Vector<decltype(DT(0) * T(0)), n> operator* (const T &b, const StaticVector<DT, ST, n> &a)
```

```
template<typename DT, typename ST, typename T, UInt n, UInt m, typename =
std::enable_if_t<is_arithmetic<T>::value>>
Matrix<decltype(DT(0) * T(0)), n, m> operator* (const StaticMatrix<DT, ST, n, m> &a, const T &b)
```

```
template<typename DT, typename ST, typename T, UInt n, UInt m, typename =
std::enable_if_t<is_arithmetic<T>::value, void>>
Matrix<decltype(DT(0) * T(0)), n, m> operator* (const T &b, const StaticMatrix<DT, ST, n, m> &a)
```

```
template<typename DT, typename ST, typename T, UInt n, typename =
std::enable_if_t<is_arithmetic<T>::value>>
```

```
SymMatrix<decltype(DT(0) * T(0)), n> operator* (const StaticSymMatrix<DT, ST, n> &a, const T &b)
```

```
template<typename DT, typename ST, typename T, UInt n, typename =  
std::enable_if_t<is_arithmetic<T>::value>>
```

```
SymMatrix<decltype(DT(0) * T(0)), n> operator* (const T &b, const StaticSymMatrix<DT, ST, n> &a)
```

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n, UInt m>  
Vector<decltype(DT1(0) * DT2(0)), n> operator* (const StaticMatrix<DT1, ST1, n, m> &a, const  
StaticVector<DT2, ST2, m> &b)
```

Matrix-vector multiplication.

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n, UInt m, UInt l>  
Matrix<decltype(DT1(0) * DT2(0)), n, m> operator* (const StaticMatrix<DT1, ST1, n, l> &a, const  
StaticMatrix<DT2, ST2, l, m> &b)
```

Matrix-matrix multiplication.

```
template<typename DT1, typename ST1, typename DT2, typename ST2, UInt n, UInt m>  
Matrix<decltype(DT1(0) * DT2(0)), n, m> outer (const StaticVector<DT1, ST1, n> &a, const  
StaticVector<DT2, ST2, m> &b)
```

```
template<typename DT, typename ST, UInt n>  
Matrix<std::remove_cv_t<DT>, n, n> dense (const StaticSymMatrix<DT, ST, n> &m)
```

```
template<typename DT, typename ST, UInt n>  
auto dense (const StaticVector<DT, ST, n> &v) -> Vector<DT, n>
```

```
template<typename DT, typename ST, UInt n>  
SymMatrix<std::remove_cv_t<DT>, n> symmetrize (const StaticMatrix<DT, ST, n, n> &m)
```

```
template<typename DT, typename ST>  
Vector<std::remove_cv_t<DT>, 3> invariants (const StaticSymMatrix<DT, ST, 3> &m)
```

```
template<typename DT, typename ST>  
Vector<std::remove_cv_t<DT>, 3> eigenvalues (const StaticSymMatrix<DT, ST, 3> &m)
```

```
void initialize (UInt num_threads = 0)  
    initialize tamaas (0 threads => let OMP_NUM_THREADS decide)
```

```
void finalize ()  
    cleanup tamaas
```

```
template<class U, class V = U>  
U exchange (U &obj, V &&new_value)  
    CUDA-compatible exchange function.
```

```
void solve (IntegralOperator::kind kind, const std::map<IntegralOperator::kind,  
    std::shared_ptr<IntegralOperator>> &operators, GridBase<Real> &input, GridBase<Real>  
    &output)
```

```
template<model_type mtype, IntegralOperator::kind otype>  
void registerWestergaardOperator (std::map<IntegralOperator::kind,  
    std::shared_ptr<IntegralOperator>> &operators, Model &model)
```

```
Real boussinesq (Real x, Real y, Real a, Real b)  
    Square patch pressure solution, Johnson (1985) page 54.
```

```

BOOST_PP_SEQ_FOR_EACH (INstantiate_Hooke, ~,
(model_type::basic_1d) (model_type::basic_2d) (model_type::surface_1d) (model_type::surface_2d))

std::ostream &operator<< (std::ostream &o, const IntegralOperator::kind &val)

std::ostream &operator<< (std::ostream &o, const Model &_this)

template<bool boundary, typename T>
std::unique_ptr<GridBase<T>> allocateGrid (Model &model)

template<bool boundary, typename T>
std::unique_ptr<GridBase<T>> allocateGrid (Model &model, UInt nc)

inline ModelDumper &operator<< (ModelDumper &dumper, Model &model)

template<typename Abstract, template<model_type> class Concrete, class ...Args>
decltype(auto) createFromModelType (model_type type, Args&&... args)

inline std::ostream &operator<< (std::ostream &o, const model_type &val)
    Print function for model_type.

template<class Function>
constexpr decltype(auto) model_type_dispatch (Function &&function, model_type type)
    Static dispatch lambda to model types.

template<class Function>
constexpr decltype(auto) dimension_dispatch (Function &&function, UInt dim)
    Static dispatch lambda to dimensions.

template<model_type type, bool boundary, typename T, template<typename, UInt> class GridType =
Grid, class Container = void>
std::unique_ptr<GridType<T, detail::dim_choice<type, boundary>::value>> allocateGrid (Container &&n,
                                                                                   UInt nc)

    Allocate a Grid unique_ptr.

template<bool boundary, typename T, typename Container>
decltype(auto) allocateGrid (model_type type, Container &&n)
    Helper function for grid allocation with model type.

template<bool boundary, typename T, typename Container>
decltype(auto) allocateGrid (model_type type, Container &&n, UInt nc)
    Helper function for grid allocation with non-standard components.

Int wrap_pbc (Int i, Int n)

template<std::size_t dim>
std::array<UInt, dim> wrap_pbc (const std::array<Int, dim> &t, const std::array<Int, dim> &n)

template<std::size_t dim>
std::array<Int, dim> cast_int (const std::array<UInt, dim> &a)

auto factorize (Grid<Real, 2> A)
    Crout(e) (Antoine... et Daniel...) factorization from https://en.wikipedia.org/wiki/Crout\_matrix\_decomposition

auto LUsubstitute (std::pair<Grid<Real, 2>, Grid<Real, 2>> LU, Grid<Real, 1> b)

```

Variables

static *complex<Real>* **dummy**

namespace **[anonymous]**

namespace **[anonymous]**

namespace **[anonymous]**

namespace **[anonymous]**

namespace **compute**

namespace **detail**

Typedefs

template<*model_type* **type**>

using **model_type_t** = std::integral_constant<*model_type*, *type*>

Convert enum value to a type.

template<*UInt* **dim**>

using **dim_t** = std::integral_constant<*UInt*, *dim*>

Convert dim value to a type.

model_types_t = std::tuple<

BOOST_PP_SEQ_ENUM(BOOST_PP_SEQ_FOR_EACH(MAKE_MODEL_TYPE, ~, (model_type::basic_1d) (model_type::basic_2d) (model_type::surface_1d) (model_type::surface_2d)))>

Enumeration of model types.

dims_t = std::tuple<

BOOST_PP_SEQ_ENUM(BOOST_PP_SEQ_FOR_EACH(MAKE_DIM_TYPE, ~, (1) (2) (3)))>

Enumeration of dimension types.

Functions

template<class **T**>

void **append_to_stream** (std::ostream &s, *T* &&head)

template<class **T**, class ...**Args**>

void **append_to_stream** (std::ostream &s, *T* &&head, *Args*&&... tail)

template<class ...**Args**>

std::string **concat_args** (*Args*&&... args)

template<bool **only_points** = false, typename ...**Grids**>

```
UInt loopSize (Grids&&... grids)
```

```
template<typename ...Sizes>
void areAllEqual (bool, std::ptrdiff_t)
```

```
template<typename ...Sizes>
void areAllEqual (bool result, std::ptrdiff_t prev, std::ptrdiff_t current)
```

```
template<typename ...Sizes>
void areAllEqual (bool result, std::ptrdiff_t prev, std::ptrdiff_t current, Sizes... rest)
```

```
template<class Function, class DynamicType, class DefaultFunction, std::size_t... Is>
constexpr decltype(auto) static_switch_dispatch (const model_types_t&, Function &&function, const
DynamicType &type, DefaultFunction
&&default_function, std::index_sequence<Is...>)
```

Specialized static dispatch for all model types.

```
template<class Function, class DynamicType, class DefaultFunction, std::size_t... Is>
constexpr decltype(auto) static_switch_dispatch (const dims_t&, Function &&function, const
DynamicType &dim, DefaultFunction
&&default_function, std::index_sequence<Is...>)
```

Specialized static dispatch for all dimensions.

```
template<class TypeTuple, class Function, class DefaultFunction, class DynamicType>
constexpr decltype(auto) tuple_dispatch_with_default (Function &&function, DefaultFunction
&&default_function, const DynamicType
&type)
```

Dispatch to tuple of types with a default case.

```
template<class TypeTuple, class Function, class DynamicType>
constexpr decltype(auto) tuple_dispatch (Function &&function, const DynamicType &type)
```

Dispatch to tuple of types, error on default.

```
namespace functional
```

```
namespace influence
```

Functions

```
template<bool conjugate, UInt dim_q>
inline Vector<Complex, dim_q + 1> computed (const VectorProxy<const Real, dim_q> &q)
```

```
template<UInt dim_q>
inline Vector<Complex, dim_q + 1> computed2 (const VectorProxy<const Real, dim_q> &q)
```

```
namespace [anonymous]
```

```
namespace iterator_
```

```
namespace mpi_dummy
```

Contains mock mpi functions.

Enums

enum class **thread** : int

Values:

enumerator **single**

enumerator **funneled**

enumerator **serialized**

enumerator **multiple**

Functions

inline bool **initialized**()

inline bool **finalized**()

inline int **init**(int*, char***)

inline int **init_thread**(int*, char***, *thread*, *thread**provided)

inline int **finalize**()

inline void **abort**(int) noexcept

inline int **rank**(*comm* = *comm::world*)

inline int **size**(*comm* = *comm::world*)

template<*operation op* = *operation::plus*, typename **T**>

inline decltype(auto) **reduce**(*T* &&val, *comm* = *comm::world*)

template<*operation op* = *operation::plus*, typename **T**>

inline decltype(auto) **allreduce**(*T* &&val, *comm* = *comm::world*)

template<typename **T**>

inline decltype(auto) **gather**(const *T**send, *T**recv, int count, *comm* = *comm::world*)

template<typename **T**>

inline decltype(auto) **scatter**(const *T**send, *T**recv, int count, *comm* = *comm::world*)

template<typename **T**>

inline decltype(auto) **scatterv**(const *T**send, const std::vector<int>&, const std::vector<int>&, *T**recv, int
recvcount, *comm* = *comm::world*)

template<typename **T**>

inline decltype(auto) **bcast**(*T**, int, *comm* = *comm::world*)

file **allocator.hh**

#include "tamaas.hh" #include "fftw/fftw_allocator.hh"

file **array.hh**

```
#include "allocator.hh" #include "errors.hh" #include "logger.hh" #include "span.hh" #include "tamaas.hh" #include <memory> #include <thrust/copy.h> #include <thrust/fill.h> #include <utility>
```

file **computes.cpp**

```
#include "computes.hh"
```

file **computes.hh**

```
#include "grid.hh" #include "loop.hh" #include "model_type.hh" #include "ranges.hh" #include "static_types.hh"
```

file **cufft_engine.cpp**

```
#include "cufft_engine.hh" #include <algorithm> #include <functional> #include <numeric>
```

file **cufft_engine.hh**

```
#include "fft_engine.hh" #include <cufft.h>
```

file **unified_allocator.hh**

```
#include "span.hh" #include <cuda_runtime_api.h> #include <memory>
```

file **errors.hh**

```
#include <sstream> #include <stdexcept>
```

Defines

TAMAAS_LOCATION

TAMAAS_MSG (...)

TAMAAS_ASSERT (cond, ...)

file **fft_engine.cpp**

```
#include "fft_engine.hh" #include "fftw/fftw_engine.hh"
```

Defines

inst (x)

file **fft_engine.hh**

```
#include "grid.hh" #include "grid_base.hh" #include "grid_hermitian.hh" #include "partitioner.hh" #include <algorithm> #include <array> #include <map> #include <memory> #include <string>
```

file **fftw_allocator.hh**

```
#include "span.hh" #include <fftw3.h> #include <memory>
```

file **fftw_engine.cpp**

```
#include "fftw_engine.hh" #include <algorithm> #include <functional> #include <numeric>
```

file **fftw_engine.hh**

```
#include "fftw_engine.hh" #include "fftw/interface.hh"
```

file **interface.hh**

```
#include "mpi/interface.hh" #include "interface_impl.hh"
```

Defines

FFTW_ESTIMATE

file **interface.hh**

```
#include "mpi_interface.hh" #include <cstdlib> #include <numeric> #include <stdexcept> #include <tuple>
```

file **interface_impl.hh**

```
#include <cstddef> #include <fftw3.h> #include <functional> #include <numeric> #include <utility>
```

file **fftw_mpi_engine.cpp**

```
#include "fftw/mpi/fftw_mpi_engine.hh" #include "fftw/interface.hh" #include "logger.hh" #include "mpi_interface.hh" #include "partitioner.hh" #include <algorithm>
```

file **fftw_mpi_engine.hh**

```
#include "fftw/fftw_engine.hh" #include "fftw/interface.hh" #include "grid.hh" #include "grid_hermitian.hh" #include <map>
```

file **grid.cpp**

```
#include "grid.hh" #include "tamaas.hh" #include <algorithm> #include <complex> #include <cstring>
```

Defines

GRID_INSTANTIATE_TYPE (type)

Class instantiation.

file **grid.hh**

```
#include "grid_base.hh" #include "tamaas.hh" #include <array> #include <numeric> #include <utility> #include <vector> #include "grid_tmpl.hh"
```

file **grid_base.hh**

```
#include "array.hh" #include "iterator.hh" #include "loop.hh" #include "mpi_interface.hh" #include "static_types.hh" #include "tamaas.hh" #include <cstddef> #include <limits> #include <utility> #include <vector>
```

Defines

VEC_OPERATOR (op)

SCALAR_OPERATOR (op)

BROADCAST_OPERATOR (op)

Broadcast operators.

SCALAR_OPERATOR_IMPL (op)

VEC_OPERATOR_IMPL (op)

BROADCAST_OPERATOR_IMPL (op)

file **grid_hermitian.cpp**

#include "grid_hermitian.hh"

Defines

GRID_HERMITIAN_INSTANCiate (type)

file **grid_hermitian.hh**

#include "grid.hh"#include "tamaas.hh"#include <complex>#include <type_traits>#include <vector>

file **grid_tmpl.hh**

#include "grid.hh"#include "tamaas.hh"

file **grid_view.hh**

#include "grid.hh"#include "tamaas.hh"#include <vector>

file **iterator.hh**

#include "tamaas.hh"#include <cstdint>#include <iterator>#include <thrust/iterator/iterator_categories.h>#include <utility>#include <vector>

file **logger.cpp**

#include "logger.hh"#include "mpi_interface.hh"#include <cstdlib>#include <iostream>#include <map>

file **logger.hh**

#include "tamaas.hh"#include <sstream>

file **loop.cpp**

#include "loop.hh"#include "tamaas.hh"

file **loop.hh**

#include "loops/apply.hh"#include "loops/loop_utils.hh"#include "mpi_interface.hh"#include "ranges.hh"#include "tamaas.hh"#include <thrust/execution_policy.h>#include <thrust/for_each.h>#include <thrust/iterator/counting_iterator.h>#include <thrust/iterator/zip_iterator.h>#include <thrust/transform_reduce.h>#include <thrust/tuple.h>#include <thrust/version.h>#include <type_traits>#include <utility>

file **apply.hh**

```
#include "tamaas.hh" #include <cstdint> #include <thrust/tuple.h> #include <utility>
```

file **loop_utils.hh**

```
#include "errors.hh" #include "tamaas.hh" #include <limits> #include <thrust/functional.h> #include <type_traits>
```

file **mpi_interface.cpp**

```
#include "mpi_interface.hh"
```

file **mpi_interface.hh**

```
#include "static_types.hh" #include "tamaas.hh" #include <cstdlib> #include <type_traits> #include <vector>
```

Defines

MPI_ERR_TOPOLOGY

file **partitioner.hh**

```
#include "fftw/interface.hh" #include "grid.hh" #include "mpi_interface.hh" #include "tamaas.hh" #include <algorithm> #include <array> #include <cstdlib> #include <functional>
```

file **ranges.hh**

```
#include "errors.hh" #include "iterator.hh" #include "mpi_interface.hh" #include "static_types.hh"
```

file **span.hh**

```
#include "tamaas.hh" #include <cstdint> #include <iterator> #include <type_traits>
```

file **static_types.hh**

```
#include "tamaas.hh" #include <thrust/detail/config/host_device.h> #include <thrust/sort.h> #include <type_traits>
```

Defines

VECTOR_OP (op)

SCALAR_OP (op)

file **statistics.cpp**

```
#include "statistics.hh" #include "fft_engine.hh" #include "loop.hh" #include "static_types.hh"
```

Variables

`std::array<UInt, dim> exponent`

file **statistics.hh**

`#include "fft_engine.hh"#include "grid.hh"`

file **tamaas.cpp**

`#include "tamaas.hh"#include "errors.hh"#include "fftw/interface.hh"#include "logger.hh"#include "mpi_interface.hh"`

Variables

`static const entry_exit_points singleton`

file **tamaas.hh**

`#include <exception>#include <iostream>#include <memory>#include <string>#include <type_traits>#include <thrust/complex.h>#include <thrust/random.h>`

Defines

TAMAAS_USE_FFTW

TAMAAS_FFTW_BACKEND_OMP

TAMAAS_FFTW_BACKEND_THREADS

TAMAAS_FFTW_BACKEND_NONE

TAMAAS_LOOP_BACKEND_OMP

TAMAAS_LOOP_BACKEND_TBB

TAMAAS_LOOP_BACKEND_CPP

TAMAAS_LOOP_BACKEND_CUDA

TAMAAS_LOOP_BACKEND

TAMAAS_FFTW_BACKEND

THRUST_DEVICE_SYSTEM

TAMAAS_ACCESSOR (var, type, name)

Convenience macros.

CUDA_LAMBDA

Cuda specific definitions.

TAMAAS_REAL_TYPE

Common types definitions.

TAMAAS_INT_TYPE

file **adhesion_functional.cpp**

#include "adhesion_functional.hh"

file **adhesion_functional.hh**

#include "functional.hh"#include <map>

file **be_engine.cpp**

#include "be_engine.hh"#include "logger.hh"#include "model.hh"

file **be_engine.hh**

#include "integral_operator.hh"#include "model_type.hh"#include "westergaard.hh"

file **boussinesq.cpp**

#include "boussinesq.hh"#include "boussinesq_helper.hh"#include "influence.hh"#include "model.hh"

file **boussinesq.hh**

#include "grid_hermitian.hh"#include "model_type.hh"#include "volume_potential.hh"

file **boussinesq_helper.hh**

#include "grid.hh"#include "grid_hermitian.hh"#include "influence.hh"#include "integration/accumulator.hh"#include "kelvin_helper.hh"#include "model.hh"#include "model_type.hh"#include <vector>

file **dcfft.cpp**

#include "dcfft.hh"#include "model.hh"#include <cmath>

file **dcfft.hh**

#include "westergaard.hh"

file **elastic_functional.cpp**

#include "elastic_functional.hh"

file **elastic_functional.hh**

#include "functional.hh"#include "model.hh"#include "tamaas.hh"

file **field_container.hh**

```
#include "grid.hh" #include "model_type.hh" #include <algorithm> #include <boost/variant.hpp> #include <memory> #include <string> #include <unordered_map>
```

file **functional.hh**

```
#include "be_engine.hh" #include "tamaas.hh"
```

file **hooke.cpp**

```
#include "hooke.hh" #include "influence.hh" #include "loop.hh" #include "model.hh" #include "static_types.hh" #include <boost/preprocessor/seq.hpp>
```

Defines

```
INstantiate_Hooke (r, _, type)
```

file **hooke.hh**

```
#include "integral_operator.hh" #include "model_type.hh"
```

file **influence.hh**

```
#include "loop.hh" #include "static_types.hh" #include <type_traits>
```

file **integral_operator.cpp**

```
#include "integral_operator.hh" #include "model.hh" #include <map> #include <ostream>
```

file **integral_operator.hh**

```
#include "field_container.hh" #include "grid_base.hh" #include "model_type.hh"
```

file **accumulator.hh**

```
#include "grid_hermitian.hh" #include "integrator.hh" #include "model_type.hh" #include "static_types.hh" #include <array> #include <thrust/iterator/zip_iterator.h> #include <vector>
```

file **element.cpp**

```
#include "element.hh"
```

file **element.hh**

```
#include "tamaas.hh" #include <expolit/expolit>
```

file **integrator.hh**

```
#include "element.hh" #include <expolit/expolit>
```

Defines

BOUNDS

file **kelvin.cpp**

```
#include "kelvin.hh" #include "logger.hh"
```

file **kelvin.hh**

```
#include "grid_hermitian.hh" #include "influence.hh" #include "integration/accumulator.hh" #include "kelvin_helper.hh" #include "model_type.hh" #include "volume_potential.hh"
```

file **kelvin_helper.hh**

```
#include "grid.hh" #include "grid_hermitian.hh" #include "influence.hh" #include "integration/accumulator.hh" #include "logger.hh" #include "model.hh" #include "model_type.hh" #include <tuple>
```

file **internal.hh**

```
#include "grid.hh" #include <memory>
```

file **isotropic_hardening.cpp**

```
#include "isotropic_hardening.hh" #include "influence.hh" #include "tamaas.hh"
```

file **isotropic_hardening.hh**

```
#include "grid.hh" #include "influence.hh" #include "internal.hh" #include "material.hh" #include "model.hh" #include "model_type.hh" #include "static_types.hh"
```

file **material.hh**

```
#include "grid.hh" #include "model.hh" #include "model_type.hh"
```

file **mfront_material.cpp**

```
#include "mfront_material.hh"
```

file **mfront_material.hh**

```
#include "material.hh" #include "model_type.hh"
```

file **meta_functional.cpp**

```
#include "meta_functional.hh"
```

file **meta_functional.hh**

```
#include "functional.hh" #include "tamaas.hh" #include <list> #include <memory>
```

file **mindlin.cpp**

```
#include "mindlin.hh" #include "boussinesq_helper.hh" #include "influence.hh" #include "kelvin_helper.hh" #include "model_type.hh"
```

file **mindlin.hh**

```
#include "grid_hermitian.hh" #include "kelvin.hh" #include "model_type.hh" #include "volume_potential.hh"
```

file **model.cpp**

```
#include "model.hh" #include "be_engine.hh" #include "logger.hh"
```

file **model.hh**

```
#include "be_engine.hh" #include "field_container.hh" #include "grid_base.hh" #include "integral_operator.hh" #include "model_dumper.hh" #include "model_type.hh" #include "tamaas.hh" #include <algorithm> #include <memory> #include <vector>
```

file **model_dumper.hh**

file **model_factory.cpp**

```
#include "model_factory.hh" #include "dcfft.hh" #include "materials/isotropic_hardening.hh" #include "model_template.hh" #include <functional>
```

Defines

CAST (derivative)

file **model_factory.hh**

```
#include "be_engine.hh" #include "model.hh" #include "model_type.hh" #include "residual.hh" #include <vector>
```

file **model_template.cpp**

```
#include "model_template.hh" #include "computes.hh" #include "hooke.hh" #include "influence.hh" #include "kelvin.hh" #include "partitioner.hh" #include "westergaard.hh"
```

Defines

CAST (derivative, ptr)

file **model_template.hh**

```
#include "grid_view.hh" #include "model.hh" #include "model_type.hh"
```

file **model_type.cpp**

```
#include "model_type.hh"
```

file **model_type.hh**

```
#include "grid.hh" #include "grid_base.hh" #include "static_types.hh" #include "tamaas.hh" #include <algorithm> #include <boost/preprocessor/cat.hpp> #include <boost/preprocessor/seq.hpp> #include <boost/preprocessor/stringize.hpp> #include <memory>
```

Defines

MODEL_TYPE_TRAITS_MACRO (type, dim, comp, bdim)

TAMAAS_MODEL_TYPES

MAKE_MODEL_TYPE (r, x, type)

MAKE_DIM_TYPE (r, x, dim)

PRINT_MODEL_TYPE (r, data, type)

SWITCH_DISPATCH_CASE (r, data, type)

SWITCH_DISPATCH_CASE (r, data, dim)

file **residual.cpp**

```
#include "residual.hh" #include "grid_view.hh" #include "model_factory.hh" #include "model_type.hh" #include <list> #include <memory>
```

file **residual.hh**

```
#include "boussinesq.hh" #include "materials/material.hh" #include "mindlin.hh" #include "model_type.hh" #include <unordered_set>
```

file **volume_potential.cpp**

```
#include "volume_potential.hh" #include "model.hh" #include <algorithm>
```

file **volume_potential.hh**

```
#include "fft_engine.hh" #include "grid_hermitian.hh" #include "grid_view.hh" #include "integral_operator.hh" #include "logger.hh" #include "model_type.hh" #include <functional>
```

file **westergaard.cpp**

```
#include "grid_hermitian.hh" #include "grid_view.hh" #include "influence.hh" #include "loop.hh" #include "model.hh" #include "model_type.hh" #include "static_types.hh" #include "westergaard.hh" #include <functional> #include <numeric>
```

file **westergaard.hh**

```
#include "fft_engine.hh" #include "grid_hermitian.hh" #include "integral_operator.hh" #include "model_type.hh" #include "tamaas.hh"
```

file **flood_fill.cpp**

```
#include "flood_fill.hh" #include "partitioner.hh" #include <algorithm> #include <limits> #include <queue>
```

file **flood_fill.hh**

```
#include "grid.hh" #include <list>
```

file **anderson.cpp**

```
#include "anderson.hh" #include <algorithm> #include <iomanip>
```

```
file anderson.hh
    #include "epic.hh"#include <deque>

file beck_teboulle.cpp
    #include "beck_teboulle.hh"#include "logger.hh"#include <iomanip>

file beck_teboulle.hh
    #include "kato.hh"

file condat.cpp
    #include "condat.hh"#include "logger.hh"#include <iomanip>

file condat.hh
    #include "kato.hh"

file contact_solver.cpp
    #include "contact_solver.hh"#include "logger.hh"#include <iomanip>#include <iostream>

file contact_solver.hh
    #include "meta_functional.hh"#include "model.hh"#include "tamaas.hh"

file dfsane_solver.cpp
    #include "dfsane_solver.hh"

file dfsane_solver.hh
    #include "ep_solver.hh"#include "grid_base.hh"#include "residual.hh"#include <deque>#include <functional>#include <utility>

file ep_solver.cpp
    #include "ep_solver.hh"#include "model_type.hh"

file ep_solver.hh
    #include "grid_base.hh"#include "residual.hh"

file epic.cpp
    #include "epic.hh"#include "logger.hh"

file epic.hh
    #include "contact_solver.hh"#include "ep_solver.hh"#include "model.hh"

file kato.cpp
    #include "kato.hh"#include "elastic_functional.hh"#include "logger.hh"#include "loop.hh"#include <iomanip>#include <iostream>#include <iterator>
```

```
file kato.hh
    #include "contact_solver.hh" #include "meta_functional.hh" #include "model_type.hh" #include
    "static_types.hh" #include "tamaas.hh"
```

```
file kato_saturated.cpp
    #include "kato_saturated.hh" #include "logger.hh" #include <iomanip> #include <limits>
```

```
file kato_saturated.hh
    #include "polonsky_keer_rey.hh" #include <limits>
```

```
file polonsky_keer_rey.cpp
    #include "polonsky_keer_rey.hh" #include "elastic_functional.hh" #include "logger.hh" #include "loop.hh" #include
    "model_type.hh" #include <iomanip>
```

```
file polonsky_keer_rey.hh
    #include "contact_solver.hh" #include "grid_view.hh" #include "meta_functional.hh" #include "westergaard.hh"
```

Defines

WESTERGAARD (type, kind, desc)

```
file polonsky_keer_tan.cpp
    #include "polonsky_keer_tan.hh" #include <iomanip>
```

```
file polonsky_keer_tan.hh
    #include "kato.hh"
```

```
file filter.hh
    #include "fft_engine.hh" #include "grid.hh" #include "grid_hermitian.hh"
```

```
file isopowerlaw.cpp
    #include "isopowerlaw.hh" #include <map>
```

```
file isopowerlaw.hh
    #include "filter.hh" #include "grid_hermitian.hh" #include "static_types.hh"
```

```
file regularized_powerlaw.cpp
    #include "regularized_powerlaw.hh"
```

```
file regularized_powerlaw.hh
    #include "filter.hh" #include "grid_hermitian.hh" #include "static_types.hh"
```

```
file surface_generator.cpp
    #include "surface_generator.hh" #include "partitioner.hh" #include <algorithm>
```

file **surface_generator.hh**

`#include "grid.hh" #include <array>`

file **surface_generator_filter.cpp**

`#include "surface_generator_filter.hh" #include <iostream>`

file **surface_generator_filter.hh**

`#include "fft_engine.hh" #include "filter.hh" #include "partitioner.hh" #include "surface_generator.hh" #include "tamaas.hh"`

file **surface_generator_random_phase.cpp**

`#include "surface_generator_random_phase.hh" #include <iostream>`

file **surface_generator_random_phase.hh**

`#include "filter.hh" #include "surface_generator_filter.hh" #include "tamaas.hh"`

file **tamaas_info.hh**

`#include <string>`

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/core**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/core/cuda**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/core/fftw**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/model/integration**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/core/loops**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/model/materials**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/model**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/core/fftw/mpi**

dir **/home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/
src/percolation**

```
dir /home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/  
src/solvers
```

```
dir /home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/src
```

```
dir /home/docs/checkouts/readthedocs.org/user_builds/tamaas/checkouts/stable/  
src/surface
```

page **index**

10.2.1 Introduction

Tamaas is a spectral-integral-equation based contact library. It is made with love to be fast and friendly!

Author

Lucas Frérot lucas.frerot@imtek.uni-freiburg.de

Author

Guillaume Anciaux guillaume.anciaux@epfl.ch

Author

Valentine Rey valentine.rey@univ-nantes.fr

Author

Son Pham-Ba son.phamba@epfl.ch

Author

Jean-François Molinari jean-francois.molinari@epfl.ch

10.2.2 License

Copyright (©) 2016-2023 EPFL (École Polytechnique Fédérale de Lausanne), Laboratory (LSMS - Laboratoire de Simulation en Mécanique des Solides) Copyright (©) 2020-2023 Lucas Frérot

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

t

- tamaas, [35](#)
- tamaas._tamaas, [35](#)
- tamaas._tamaas.compute, [59](#)
- tamaas._tamaas.mpi, [59](#)
- tamaas.dumpers, [60](#)
- tamaas.nonlinear_solvers, [63](#)
- tamaas.utils, [64](#)

Symbols

- `_DFSANESolver` (class in `tamaas._tamaas`), 58
- `__init__()` (`tamaas._tamaas.AdhesionFunctional` method), 35
- `__init__()` (`tamaas._tamaas.AndersonMixing` method), 35
- `__init__()` (`tamaas._tamaas.AssertionError` method), 36
- `__init__()` (`tamaas._tamaas.BEEngine` method), 36
- `__init__()` (`tamaas._tamaas.BeckTeboulle` method), 36
- `__init__()` (`tamaas._tamaas.Cluster1D` method), 37
- `__init__()` (`tamaas._tamaas.Cluster2D` method), 37
- `__init__()` (`tamaas._tamaas.Cluster3D` method), 38
- `__init__()` (`tamaas._tamaas.Condat` method), 38
- `__init__()` (`tamaas._tamaas.ContactSolver` method), 39
- `__init__()` (`tamaas._tamaas.EPICSolver` method), 40
- `__init__()` (`tamaas._tamaas.EPSolver` method), 40
- `__init__()` (`tamaas._tamaas.ElasticFunctionalGap` method), 40
- `__init__()` (`tamaas._tamaas.ElasticFunctionalPressure` method), 40
- `__init__()` (`tamaas._tamaas.ExponentialAdhesion-Functional` method), 41
- `__init__()` (`tamaas._tamaas.FieldContainer` method), 41
- `__init__()` (`tamaas._tamaas.Filter1D` method), 41
- `__init__()` (`tamaas._tamaas.Filter2D` method), 41
- `__init__()` (`tamaas._tamaas.FloatingPointError` method), 42
- `__init__()` (`tamaas._tamaas.FloodFill` method), 42
- `__init__()` (`tamaas._tamaas.Functional` method), 42
- `__init__()` (`tamaas._tamaas.IntegralOperator` method), 42
- `__init__()` (`tamaas._tamaas.Isopowerlaw1D` method), 43
- `__init__()` (`tamaas._tamaas.Isopowerlaw2D` method), 43
- `__init__()` (`tamaas._tamaas.Kato` method), 44
- `__init__()` (`tamaas._tamaas.KatoSaturated` method), 45
- `__init__()` (`tamaas._tamaas.KatoSaturated.type` method), 46
- `__init__()` (`tamaas._tamaas.LogLevel` method), 46
- `__init__()` (`tamaas._tamaas.Logger` method), 46
- `__init__()` (`tamaas._tamaas.MaugisAdhesionFunctional` method), 46
- `__init__()` (`tamaas._tamaas.Model` method), 47
- `__init__()` (`tamaas._tamaas.ModelDumper` method), 49
- `__init__()` (`tamaas._tamaas.ModelFactory` method), 49
- `__init__()` (`tamaas._tamaas.ModelTypeError` method), 49
- `__init__()` (`tamaas._tamaas.NotImplementedError` method), 50
- `__init__()` (`tamaas._tamaas.PolonskyKeerRey` method), 50
- `__init__()` (`tamaas._tamaas.PolonskyKeerRey.type` method), 51
- `__init__()` (`tamaas._tamaas.PolonskyKeerTan` method), 51
- `__init__()` (`tamaas._tamaas.RegularizedPowerlaw1D` method), 51
- `__init__()` (`tamaas._tamaas.RegularizedPowerlaw2D` method), 52
- `__init__()` (`tamaas._tamaas.Residual` method), 52
- `__init__()` (`tamaas._tamaas.SquaredExponentialAdhesionFunctional` method), 53
- `__init__()` (`tamaas._tamaas.Statistics1D` method), 53
- `__init__()` (`tamaas._tamaas.Statistics2D` method), 54
- `__init__()` (`tamaas._tamaas.SurfaceGenerator1D` method), 54
- `__init__()` (`tamaas._tamaas.SurfaceGenerator2D` method), 54
- `__init__()` (`tamaas._tamaas.SurfaceGeneratorFilter1D` method), 55
- `__init__()` (`tamaas._tamaas.SurfaceGeneratorFilter2D` method), 55
- `__init__()` (`tamaas._tamaas.SurfaceGeneratorRandomPhase1D` method), 56
- `__init__()` (`tamaas._tamaas.SurfaceGeneratorRandomPhase2D` method), 56

`__init__()` (*tamaas._tamaas.TamaasInfo* method), 57
`__init__()` (*tamaas._tamaas._DFSANESolver* method), 58
`__init__()` (*tamaas._tamaas.integration_method* method), 58
`__init__()` (*tamaas._tamaas.model_type* method), 58
`__init__()` (*tamaas._tamaas.mpi.sequential* method), 59
`__init__()` (*tamaas.dumpers.FieldDumper* method), 60
`__init__()` (*tamaas.dumpers.H5Dumper* method), 61
`__init__()` (*tamaas.dumpers.JSONDumper* method), 60
`__init__()` (*tamaas.dumpers.NumpyDumper* method), 61
`__init__()` (*tamaas.dumpers.UVWDumper* method), 62
`__init__()` (*tamaas.dumpers.UVWGroupDumper* method), 62
`__init__()` (*tamaas.nonlinear_solvers.DFSANESolver* method), 63
`__init__()` (*tamaas.nonlinear_solvers.NLNoConvergence* method), 63
`__init__()` (*tamaas.nonlinear_solvers.NewtonKrylov-Solver* method), 64

A

`acceleratedSolve()` (*tamaas._tamaas.Anderson-Mixing* method), 36
`acceleratedSolve()` (*tamaas._tamaas.EPICSolver* method), 40
`add_field()` (*tamaas.dumpers.FieldDumper* method), 60
`add_field()` (*tamaas.dumpers.H5Dumper* method), 61
`add_field()` (*tamaas.dumpers.NumpyDumper* method), 61
`add_field()` (*tamaas.dumpers.UVWDumper* method), 62
`add_field()` (*tamaas.dumpers.UVWGroupDumper* method), 62
`addDumper()` (*tamaas._tamaas.Model* method), 47
`addFunctionalTerm()` (*tamaas._tamaas.BeckTeboulle* method), 36
`addFunctionalTerm()` (*tamaas._tamaas.Condat* method), 38
`addFunctionalTerm()` (*tamaas._tamaas.Contact-Solver* method), 39
`addFunctionalTerm()` (*tamaas._tamaas.Kato* method), 44
`addFunctionalTerm()` (*tamaas._tamaas.KatoSaturated* method), 45
`addFunctionalTerm()` (*tamaas._tamaas.Polonsky-KeerKey* method), 50

`addFunctionalTerm()` (*tamaas._tamaas.Polonsky-KeerTan* method), 51
`AdhesionFunctional` (class in *tamaas._tamaas*), 35
`alpha()` (*tamaas._tamaas.Isopowerlaw1D* method), 43
`alpha()` (*tamaas._tamaas.Isopowerlaw2D* method), 43
`AndersonMixing` (class in *tamaas._tamaas*), 35
`apply()` (*tamaas._tamaas.IntegralOperator* method), 42
`applyElasticity()` (*tamaas._tamaas.Model* method), 47
`applyTangent()` (*tamaas._tamaas.Residual* method), 52
`area` (*tamaas._tamaas.Cluster1D* property), 37
`area` (*tamaas._tamaas.Cluster2D* property), 37
`area` (*tamaas._tamaas.Cluster3D* property), 38
`args` (*tamaas._tamaas.AssertionError* attribute), 36
`args` (*tamaas._tamaas.FloatingPointError* attribute), 42
`args` (*tamaas._tamaas.ModelTypeError* attribute), 49
`args` (*tamaas._tamaas.NotImplementedError* attribute), 50
`args` (*tamaas.nonlinear_solvers.NLNoConvergence* attribute), 63
`AssertionError`, 36

B

`backend` (*tamaas._tamaas.TamaasInfo* attribute), 57
`basic_1d` (*tamaas._tamaas.model_type* attribute), 58
`basic_2d` (*tamaas._tamaas.model_type* attribute), 58
`be_engine` (*tamaas._tamaas.Model* property), 47
`BeckTeboulle` (class in *tamaas._tamaas*), 36
`BEEngine` (class in *tamaas._tamaas*), 36
`beforeSolve()` (*tamaas._tamaas._DFSANESolver* method), 59
`beforeSolve()` (*tamaas._tamaas.EPSolver* method), 40
`beforeSolve()` (*tamaas.nonlinear_solvers.DFSANE-Solver* method), 63
`beforeSolve()` (*tamaas.nonlinear_solvers.NewtonKrylovSolver* method), 64
`boundary_fields` (*tamaas._tamaas.Model* property), 47
`boundary_shape` (*tamaas._tamaas.Model* property), 47
`boundary_system_size` (*tamaas._tamaas.Model* property), 47
`bounding_box` (*tamaas._tamaas.Cluster1D* property), 37
`bounding_box` (*tamaas._tamaas.Cluster2D* property), 37
`bounding_box` (*tamaas._tamaas.Cluster3D* property), 38
`BOUNDS` (*C* macro), 182
`branch` (*tamaas._tamaas.TamaasInfo* attribute), 57
`BROADCAST_OPERATOR` (*C* macro), 177
`BROADCAST_OPERATOR_IMPL` (*C* macro), 177

`build_type(tamaas._tamaas.TamaasInfo attribute)`, 57
`buildSurface()` (`tamaas._tamaas.SurfaceGenerator1D` method), 54
`buildSurface()` (`tamaas._tamaas.SurfaceGenerator2D` method), 54
`buildSurface()` (`tamaas._tamaas.SurfaceGeneratorFilter1D` method), 55
`buildSurface()` (`tamaas._tamaas.SurfaceGeneratorFilter2D` method), 55
`buildSurface()` (`tamaas._tamaas.SurfaceGeneratorRandomPhase1D` method), 56
`buildSurface()` (`tamaas._tamaas.SurfaceGeneratorRandomPhase2D` method), 57

C

`CAST (C macro)`, 183
`Cluster1D (class in tamaas._tamaas)`, 37
`Cluster2D (class in tamaas._tamaas)`, 37
`Cluster3D (class in tamaas._tamaas)`, 38
`commit(tamaas._tamaas.TamaasInfo attribute)`, 57
`computeAutocorrelation()` (`tamaas._tamaas.Statistics1D` static method), 53
`computeAutocorrelation()` (`tamaas._tamaas.Statistics2D` static method), 54
`computeCost()` (`tamaas._tamaas.BeckTeboulle` method), 36
`computeCost()` (`tamaas._tamaas.Condat` method), 38
`computeCost()` (`tamaas._tamaas.Kato` method), 44
`computeCost()` (`tamaas._tamaas.PolonskyKeerTan` method), 51
`computeError()` (`tamaas._tamaas.KatoSaturated` method), 45
`computeError()` (`tamaas._tamaas.PolonskyKeerRey` method), 50
`computeF()` (`tamaas._tamaas.AdhesionFunctional` method), 35
`computeF()` (`tamaas._tamaas.ElasticFunctionalGap` method), 40
`computeF()` (`tamaas._tamaas.ElasticFunctionalPressure` method), 41
`computeF()` (`tamaas._tamaas.ExponentialAdhesionFunctional` method), 41
`computeF()` (`tamaas._tamaas.Functional` method), 42
`computeF()` (`tamaas._tamaas.MaugisAdhesionFunctional` method), 46
`computeF()` (`tamaas._tamaas.SquaredExponentialAdhesionFunctional` method), 53
`computeFDRMSSlope()` (`tamaas._tamaas.Statistics1D` static method), 53
`computeFDRMSSlope()` (`tamaas._tamaas.Statistics2D` static method), 54

`computeFilter()` (`tamaas._tamaas.Filter1D` method), 41
`computeFilter()` (`tamaas._tamaas.Filter2D` method), 41
`computeFilter()` (`tamaas._tamaas.Isopowerlaw1D` method), 43
`computeFilter()` (`tamaas._tamaas.Isopowerlaw2D` method), 43
`computeFilter()` (`tamaas._tamaas.RegularizedPowerlaw1D` method), 52
`computeFilter()` (`tamaas._tamaas.RegularizedPowerlaw2D` method), 52
`computeGradF()` (`tamaas._tamaas.AdhesionFunctional` method), 35
`computeGradF()` (`tamaas._tamaas.ElasticFunctionalGap` method), 40
`computeGradF()` (`tamaas._tamaas.ElasticFunctionalPressure` method), 41
`computeGradF()` (`tamaas._tamaas.ExponentialAdhesionFunctional` method), 41
`computeGradF()` (`tamaas._tamaas.Functional` method), 42
`computeGradF()` (`tamaas._tamaas.MaugisAdhesionFunctional` method), 46
`computeGradF()` (`tamaas._tamaas.SquaredExponentialAdhesionFunctional` method), 53
`computeMoments()` (`tamaas._tamaas.Statistics1D` static method), 53
`computeMoments()` (`tamaas._tamaas.Statistics2D` static method), 54
`computePowerSpectrum()` (`tamaas._tamaas.Statistics1D` static method), 53
`computePowerSpectrum()` (`tamaas._tamaas.Statistics2D` static method), 54
`computeResidual()` (`tamaas._tamaas.Residual` method), 52
`computeResidualDisplacement()` (`tamaas._tamaas.Residual` method), 52
`computeRMSHeights()` (`tamaas._tamaas.Statistics1D` static method), 53
`computeRMSHeights()` (`tamaas._tamaas.Statistics2D` static method), 54
`computeSpectralRMSSlope()` (`tamaas._tamaas.Statistics1D` static method), 53
`computeSpectralRMSSlope()` (`tamaas._tamaas.Statistics2D` static method), 54
`Condat (class in tamaas._tamaas)`, 38
`contact()` (`tamaas._tamaas.Statistics1D` static method), 53
`contact()` (`tamaas._tamaas.Statistics2D` static method), 54
`ContactSolver (class in tamaas._tamaas)`, 39

`createModel()` (*tamaas._tamaas.ModelFactory static method*), 49
`createResidual()` (*tamaas._tamaas.ModelFactory static method*), 49
`CUDA_LAMBDA` (*C macro*), 180
`cufft` (*C++ type*), 164
`cufft::plan` (*C++ struct*), 138
`cufft::plan::_plan` (*C++ member*), 139
`cufft::plan::~~plan` (*C++ function*), 138
`cufft::plan::operator cufftHandle` (*C++ function*), 138
`cufft::plan::operator=` (*C++ function*), 138
`cufft::plan::plan` (*C++ function*), 138
`cutoff` (*tamaas._tamaas.integration_method attribute*), 58

D

`debug` (*tamaas._tamaas.LogLevel attribute*), 46
`deviatoric()` (*in module tamaas._tamaas.compute*), 59
`DFSANECXXSolver` (*in module tamaas.nonlinear_solvers*), 64
`DFSANESolver` (*class in tamaas.nonlinear_solvers*), 63
`diff` (*tamaas._tamaas.TamaasInfo attribute*), 57
`dirac` (*tamaas._tamaas.IntegralOperator attribute*), 42
`dirichlet` (*tamaas._tamaas.IntegralOperator attribute*), 42
`displacement` (*tamaas._tamaas.Model property*), 47
`dump()` (*tamaas._tamaas.Model method*), 47
`dump()` (*tamaas._tamaas.ModelDumper method*), 49
`dump()` (*tamaas.dumpers.FieldDumper method*), 60
`dump()` (*tamaas.dumpers.H5Dumper method*), 61
`dump()` (*tamaas.dumpers.JSONDumper method*), 60
`dump()` (*tamaas.dumpers.NumpyDumper method*), 61
`dump()` (*tamaas.dumpers.UVWDumper method*), 62
`dump()` (*tamaas.dumpers.UVWGroupDumper method*), 62
`dump_freq` (*tamaas._tamaas.BeckTeboulle property*), 36
`dump_freq` (*tamaas._tamaas.Condat property*), 38
`dump_freq` (*tamaas._tamaas.ContactSolver property*), 39
`dump_freq` (*tamaas._tamaas.Kato property*), 44
`dump_freq` (*tamaas._tamaas.KatoSaturated property*), 45
`dump_freq` (*tamaas._tamaas.PolonskyKeerRey property*), 50
`dump_freq` (*tamaas._tamaas.PolonskyKeerTan property*), 51

E

`E` (*tamaas._tamaas.Model property*), 47
`E_star` (*tamaas._tamaas.Model property*), 47
`eigenvalues()` (*in module tamaas._tamaas.compute*), 59

`elasticEnergy()` (*tamaas._tamaas.Isopowerlaw1D method*), 43
`elasticEnergy()` (*tamaas._tamaas.Isopowerlaw2D method*), 44
`ElasticFunctionalGap` (*class in tamaas._tamaas*), 40
`ElasticFunctionalPressure` (*class in tamaas._tamaas*), 40
`EPICSolver` (*class in tamaas._tamaas*), 39
`EPSolver` (*class in tamaas._tamaas*), 40
`error` (*tamaas._tamaas.LogLevel attribute*), 46
`exponent` (*C++ member*), 179
`ExponentialAdhesionFunctional` (*class in tamaas._tamaas*), 41
`extension` (*tamaas.dumpers.FieldDumper attribute*), 60
`extension` (*tamaas.dumpers.H5Dumper attribute*), 61
`extension` (*tamaas.dumpers.NumpyDumper attribute*), 61
`extension` (*tamaas.dumpers.UVWDumper attribute*), 62
`extension` (*tamaas.dumpers.UVWGroupDumper attribute*), 62
`extent` (*tamaas._tamaas.Cluster1D property*), 37
`extent` (*tamaas._tamaas.Cluster2D property*), 37
`extent` (*tamaas._tamaas.Cluster3D property*), 38

F

`fftw` (*C++ type*), 164
`FFTW_ESTIMATE` (*C macro*), 176
`fftw_impl` (*C++ type*), 164
`fftw_impl::cleanup_threads` (*C++ function*), 164
`fftw_impl::destroy` (*C++ function*), 165
`fftw_impl::execute` (*C++ function*), 164, 165
`fftw_impl::free` (*C++ function*), 164
`fftw_impl::helper` (*C++ struct*), 103
`fftw_impl::helper<double>` (*C++ struct*), 103
`fftw_impl::helper<double>::alloc_complex` (*C++ function*), 104
`fftw_impl::helper<double>::alloc_real` (*C++ function*), 104
`fftw_impl::helper<double>::complex` (*C++ type*), 103
`fftw_impl::helper<double>::plan` (*C++ type*), 103
`fftw_impl::helper<float>` (*C++ struct*), 104
`fftw_impl::helper<float>::alloc_complex` (*C++ function*), 104
`fftw_impl::helper<float>::alloc_real` (*C++ function*), 104
`fftw_impl::helper<float>::complex` (*C++ type*), 104
`fftw_impl::helper<float>::plan` (*C++ type*), 104

fftw_impl::helper<long double> (C++ struct), 104
 fftw_impl::helper<long double>::alloc_complex (C++ function), 104
 fftw_impl::helper<long double>::alloc_real (C++ function), 104
 fftw_impl::helper<long double>::complex (C++ type), 104
 fftw_impl::helper<long double>::plan (C++ type), 104
 fftw_impl::init_threads (C++ function), 164
 fftw_impl::mpi_dummy (C++ type), 165
 fftw_impl::mpi_dummy::cleanup (C++ function), 166
 fftw_impl::mpi_dummy::init (C++ function), 166
 fftw_impl::mpi_dummy::local_size_many (C++ function), 166
 fftw_impl::plan (C++ struct), 139
 fftw_impl::plan::_plan (C++ member), 139
 fftw_impl::plan::~~plan (C++ function), 139
 fftw_impl::plan::operator typename helper<T>::plan (C++ function), 139
 fftw_impl::plan::operator= (C++ function), 139
 fftw_impl::plan::plan (C++ function), 139
 fftw_impl::plan_1d_backward (C++ function), 164, 165
 fftw_impl::plan_1d_forward (C++ function), 164, 165
 fftw_impl::plan_2d_backward (C++ function), 164, 165
 fftw_impl::plan_2d_forward (C++ function), 164, 165
 fftw_impl::plan_many_backward (C++ function), 164, 165
 fftw_impl::plan_many_forward (C++ function), 164, 165
 fftw_impl::plan_with_nthreads (C++ function), 164
 fftw_impl::ptr (C++ struct), 142
 fftw_impl::ptr::_ptr (C++ member), 143
 fftw_impl::ptr::~~ptr (C++ function), 143
 fftw_impl::ptr::operator T* (C++ function), 143
 FieldContainer (class in *tamaas._tamaas*), 41
 FieldDumper (class in *tamaas.dumpers*), 60
 file_path (*tamaas.dumpers.FieldDumper* property), 61
 file_path (*tamaas.dumpers.H5Dumper* property), 61
 file_path (*tamaas.dumpers.NumpyDumper* property), 61
 file_path (*tamaas.dumpers.UVWDumper* property), 62
 file_path (*tamaas.dumpers.UVWGroupDumper* property), 62
 Filter1D (class in *tamaas._tamaas*), 41
 Filter2D (class in *tamaas._tamaas*), 41
 finalize () (in module *tamaas._tamaas*), 57
 FloatingPointError, 41
 FloodFill (class in *tamaas._tamaas*), 42
 from_voigt () (in module *tamaas._tamaas.compute*), 60
 Functional (class in *tamaas._tamaas*), 42
 functional (*tamaas._tamaas.BeckTeboulle* property), 37
 functional (*tamaas._tamaas.Condat* property), 38
 functional (*tamaas._tamaas.ContactSolver* property), 39
 functional (*tamaas._tamaas.Kato* property), 44
 functional (*tamaas._tamaas.KatoSaturated* property), 45
 functional (*tamaas._tamaas.PolonskyKeerRey* property), 50
 functional (*tamaas._tamaas.PolonskyKeerTan* property), 51
G
 gap (*tamaas._tamaas.KatoSaturated* attribute), 45
 gap (*tamaas._tamaas.KatoSaturated.type* attribute), 46
 gap (*tamaas._tamaas.PolonskyKeerRey* attribute), 50
 gap (*tamaas._tamaas.PolonskyKeerRey.type* attribute), 51
 gather () (in module *tamaas._tamaas.mpi*), 59
 get () (*tamaas._tamaas.Logger* method), 46
 get_fields () (*tamaas.dumpers.FieldDumper* method), 60
 get_fields () (*tamaas.dumpers.H5Dumper* method), 61
 get_fields () (*tamaas.dumpers.NumpyDumper* method), 61
 get_fields () (*tamaas.dumpers.UVWDumper* method), 62
 get_fields () (*tamaas.dumpers.UVWGroupDumper* method), 62
 get_log_level () (in module *tamaas._tamaas*), 57
 getArea () (*tamaas._tamaas.Cluster1D* method), 37
 getArea () (*tamaas._tamaas.Cluster2D* method), 38
 getArea () (*tamaas._tamaas.Cluster3D* method), 38
 getBEngine () (*tamaas._tamaas.Model* method), 47
 getBoundaryDiscretization () (*tamaas._tamaas.Model* method), 47
 getBoundarySystemSize () (*tamaas._tamaas.Model* method), 47
 getClusters () (*tamaas._tamaas.FloodFill* static method), 42
 getDiscretization () (*tamaas._tamaas.Model* method), 47
 getDisplacement () (*tamaas._tamaas.Model* method), 48

`getField()` (*tamaas._tamaas.FieldContainer method*), 41

`getField()` (*tamaas._tamaas.Model method*), 48

`getFields()` (*tamaas._tamaas.FieldContainer method*), 41

`getFields()` (*tamaas._tamaas.Model method*), 48

`getHertzModulus()` (*tamaas._tamaas.Model method*), 48

`getIntegralOperator()` (*tamaas._tamaas.Model method*), 48

`getKind()` (*tamaas._tamaas.IntegralOperator method*), 42

`getModel()` (*tamaas._tamaas.BEEngine method*), 36

`getModel()` (*tamaas._tamaas.IntegralOperator method*), 42

`getPerimeter()` (*tamaas._tamaas.Cluster1D method*), 37

`getPerimeter()` (*tamaas._tamaas.Cluster2D method*), 38

`getPerimeter()` (*tamaas._tamaas.Cluster3D method*), 38

`getPoints()` (*tamaas._tamaas.Cluster1D method*), 37

`getPoints()` (*tamaas._tamaas.Cluster2D method*), 38

`getPoints()` (*tamaas._tamaas.Cluster3D method*), 38

`getPoissonRatio()` (*tamaas._tamaas.Model method*), 48

`getResidual()` (*tamaas._tamaas.DFSANESolver method*), 59

`getResidual()` (*tamaas._tamaas.EPSolver method*), 40

`getResidual()` (*tamaas.nonlinear_solvers.DFSANE-Solver method*), 63

`getResidual()` (*tamaas.nonlinear_solvers.NewtonKrylovSolver method*), 64

`getSegments()` (*tamaas._tamaas.FloodFill static method*), 42

`getShearModulus()` (*tamaas._tamaas.Model method*), 48

`getStrainIncrement()` (*tamaas._tamaas.DFSANESolver method*), 59

`getStrainIncrement()` (*tamaas._tamaas.EPSolver method*), 40

`getStrainIncrement()` (*tamaas.nonlinear_solvers.DFSANESolver method*), 63

`getStrainIncrement()` (*tamaas.nonlinear_solvers.NewtonKrylovSolver method*), 64

`getStress()` (*tamaas._tamaas.Residual method*), 52

`getSystemSize()` (*tamaas._tamaas.Model method*), 48

`getTraction()` (*tamaas._tamaas.Model method*), 48

`getType()` (*tamaas._tamaas.IntegralOperator method*), 42

`getVector()` (*tamaas._tamaas.Residual method*), 52

`getVolumes()` (*tamaas._tamaas.FloodFill static method*), 42

`getYoungModulus()` (*tamaas._tamaas.Model method*), 48

`global_shape` (*tamaas._tamaas.Model property*), 48

`global_shape()` (*in module tamaas._tamaas.mpi*), 59

`graphArea()` (*tamaas._tamaas.Statistics1D static method*), 53

`graphArea()` (*tamaas._tamaas.Statistics2D static method*), 54

`GRID_HERMITIAN_INSTANCiate` (*C macro*), 177

`GRID_INSTANCiate_TYPE` (*C macro*), 176

H

`H5Dumper` (*class in tamaas.dumpers*), 61

`has_mpi` (*tamaas._tamaas.TamaasInfo attribute*), 57

`hertz_surface()` (*in module tamaas.utils*), 65

`hurst` (*tamaas._tamaas.Isopowerlaw1D property*), 43

`hurst` (*tamaas._tamaas.Isopowerlaw2D property*), 44

`hurst` (*tamaas._tamaas.RegularizedPowerlaw1D property*), 52

`hurst` (*tamaas._tamaas.RegularizedPowerlaw2D property*), 52

I

`info` (*tamaas._tamaas.LogLevel attribute*), 46

`initialize()` (*in module tamaas._tamaas*), 57

`inst` (*C macro*), 175

`INSTANTIATE_HOOKE` (*C macro*), 181

`IntegralOperator` (*class in tamaas._tamaas*), 42

`integration_method` (*class in tamaas._tamaas*), 57

`Isopowerlaw1D` (*class in tamaas._tamaas*), 43

`Isopowerlaw2D` (*class in tamaas._tamaas*), 43

J

`JSONDumper` (*class in tamaas.dumpers*), 60

K

`Kato` (*class in tamaas._tamaas*), 44

`KatoSaturated` (*class in tamaas._tamaas*), 45

`KatoSaturated.type` (*class in tamaas._tamaas*), 45

`kind` (*tamaas._tamaas.IntegralOperator property*), 42

L

`linear` (*tamaas._tamaas.integration_method attribute*), 58

`load_path()` (*in module tamaas.utils*), 64

`local_offset()` (*in module tamaas._tamaas.mpi*), 59

`local_shape()` (*in module tamaas._tamaas.mpi*), 59

`log_context()` (*in module tamaas.utils*), 64

`Logger` (*class in tamaas._tamaas*), 46

`LogLevel` (*class in tamaas._tamaas*), 46

M

MAKE_DIM_TYPE (C macro), 184
 MAKE_MODEL_TYPE (C macro), 184
 material (*tamaas._tamaas.Residual* property), 52
 matvec() (*tamaas._tamaas.IntegralOperator* method), 42
 MaugisAdhesionFunctional (class in *tamaas._tamaas*), 46
 max_iter (*tamaas._tamaas.DFSANESolver* property), 59
 max_iter (*tamaas._tamaas.AndersonMixing* property), 36
 max_iter (*tamaas._tamaas.BeckTeboulle* property), 37
 max_iter (*tamaas._tamaas.Condat* property), 39
 max_iter (*tamaas._tamaas.ContactSolver* property), 39
 max_iter (*tamaas._tamaas.EPICSolver* property), 40
 max_iter (*tamaas._tamaas.Kato* property), 44
 max_iter (*tamaas._tamaas.KatoSaturated* property), 45
 max_iter (*tamaas._tamaas.PolonskyKeerRey* property), 50
 max_iter (*tamaas._tamaas.PolonskyKeerTan* property), 51
 Model (class in *tamaas._tamaas*), 47
 model (*tamaas._tamaas.AndersonMixing* property), 36
 model (*tamaas._tamaas.BeckTeboulle* property), 37
 model (*tamaas._tamaas.BEEngine* property), 36
 model (*tamaas._tamaas.Condat* property), 39
 model (*tamaas._tamaas.ContactSolver* property), 39
 model (*tamaas._tamaas.EPICSolver* property), 40
 model (*tamaas._tamaas.IntegralOperator* property), 42
 model (*tamaas._tamaas.Kato* property), 44
 model (*tamaas._tamaas.KatoSaturated* property), 45
 model (*tamaas._tamaas.PolonskyKeerRey* property), 50
 model (*tamaas._tamaas.PolonskyKeerTan* property), 51
 model (*tamaas._tamaas.Residual* property), 53
 model_type (class in *tamaas._tamaas*), 58
 MODEL_TYPE_TRAITS_MACRO (C macro), 184
 ModelDumper (class in *tamaas._tamaas*), 48
 ModelFactory (class in *tamaas._tamaas*), 49
 ModelTypeError, 49
 module
 tamaas, 35
 tamaas._tamaas, 35
 tamaas._tamaas.compute, 59
 tamaas._tamaas.mpi, 59
 tamaas.dumpers, 60
 tamaas.nonlinear_solvers, 63
 tamaas.utils, 64
 moments() (*tamaas._tamaas.Isopowerlaw1D* method), 43
 moments() (*tamaas._tamaas.Isopowerlaw2D* method), 44
 MPI_ERR_TOPOLOGY (C macro), 178
 mu (*tamaas._tamaas.Model* property), 48

N

name (*tamaas._tamaas.integration_method* property), 58
 name (*tamaas._tamaas.KatoSaturated.type* property), 46
 name (*tamaas._tamaas.LogLevel* property), 46
 name (*tamaas._tamaas.model_type* property), 58
 name (*tamaas._tamaas.PolonskyKeerRey.type* property), 51
 name_format (*tamaas.dumpers.FieldDumper* attribute), 60
 name_format (*tamaas.dumpers.H5Dumper* attribute), 61
 name_format (*tamaas.dumpers.NumpyDumper* attribute), 61
 name_format (*tamaas.dumpers.UVWDumper* attribute), 62
 name_format (*tamaas.dumpers.UVWGroupDumper* attribute), 62
 neumann (*tamaas._tamaas.IntegralOperator* attribute), 42
 NewtonKrylovSolver (class in *tamaas.nonlinear_solvers*), 64
 NLNoConvergence, 63
 NotImplementedError, 49
 nu (*tamaas._tamaas.Model* property), 48
 NumpyDumper (class in *tamaas.dumpers*), 61

O

operators (*tamaas._tamaas.Model* property), 48

P

parameters (*tamaas._tamaas.AdhesionFunctional* property), 35
 parameters (*tamaas._tamaas.ExponentialAdhesionFunctional* property), 41
 parameters (*tamaas._tamaas.MaugisAdhesionFunctional* property), 47
 parameters (*tamaas._tamaas.SquaredExponentialAdhesionFunctional* property), 53
 perimeter (*tamaas._tamaas.Cluster1D* property), 37
 perimeter (*tamaas._tamaas.Cluster2D* property), 38
 perimeter (*tamaas._tamaas.Cluster3D* property), 38
 pmax (*tamaas._tamaas.KatoSaturated* property), 45
 points (*tamaas._tamaas.Cluster1D* property), 37
 points (*tamaas._tamaas.Cluster2D* property), 38
 points (*tamaas._tamaas.Cluster3D* property), 38
 PolonskyKeerRey (class in *tamaas._tamaas*), 50
 PolonskyKeerRey.type (class in *tamaas._tamaas*), 51
 PolonskyKeerTan (class in *tamaas._tamaas*), 51
 pressure (*tamaas._tamaas.KatoSaturated* attribute), 45
 pressure (*tamaas._tamaas.KatoSaturated.type* attribute), 46
 pressure (*tamaas._tamaas.PolonskyKeerRey* attribute), 50

pressure (tamaas._tamaas.PolonskyKeerRey.type attribute), 51
PRINT_MODEL_TYPE (C macro), 184
publications() (in module tamaas.utils), 64

Q

q0 (tamaas._tamaas.Isopowerlaw1D property), 43
q0 (tamaas._tamaas.Isopowerlaw2D property), 44
q1 (tamaas._tamaas.Isopowerlaw1D property), 43
q1 (tamaas._tamaas.Isopowerlaw2D property), 44
q1 (tamaas._tamaas.RegularizedPowerlaw1D property), 52
q1 (tamaas._tamaas.RegularizedPowerlaw2D property), 52
q2 (tamaas._tamaas.Isopowerlaw1D property), 43
q2 (tamaas._tamaas.Isopowerlaw2D property), 44
q2 (tamaas._tamaas.RegularizedPowerlaw1D property), 52
q2 (tamaas._tamaas.RegularizedPowerlaw2D property), 52

R

radial_average() (in module tamaas.utils), 65
radialPSDMoment() (tamaas._tamaas.Isopowerlaw1D method), 43
radialPSDMoment() (tamaas._tamaas.Isopowerlaw2D method), 44
random_seed (tamaas._tamaas.SurfaceGenerator1D property), 54
random_seed (tamaas._tamaas.SurfaceGenerator2D property), 54
random_seed (tamaas._tamaas.SurfaceGeneratorFilter1D property), 55
random_seed (tamaas._tamaas.SurfaceGeneratorFilter2D property), 55
random_seed (tamaas._tamaas.SurfaceGeneratorRandomPhase1D property), 56
random_seed (tamaas._tamaas.SurfaceGeneratorRandomPhase2D property), 57
rank() (in module tamaas._tamaas.mpi), 59
read() (tamaas.dumpers.FieldDumper class method), 60
read() (tamaas.dumpers.H5Dumper class method), 61
read() (tamaas.dumpers.JSONDumper class method), 60
read() (tamaas.dumpers.NumpyDumper class method), 61
read() (tamaas.dumpers.UVWDumper class method), 62
read() (tamaas.dumpers.UVWGroupDumper class method), 62
read_sequence() (tamaas.dumpers.FieldDumper class method), 60
read_sequence() (tamaas.dumpers.H5Dumper class method), 62
read_sequence() (tamaas.dumpers.NumpyDumper class method), 61

read_sequence() (tamaas.dumpers.UVWDumper class method), 62
read_sequence() (tamaas.dumpers.UVWGroupDumper class method), 63
registerDirichlet() (tamaas._tamaas.BEEngine method), 36
registerField() (tamaas._tamaas.FieldContainer method), 41
registerField() (tamaas._tamaas.Model method), 48
registerHookeField() (tamaas._tamaas.ModelFactory static method), 49
registerNeumann() (tamaas._tamaas.BEEngine method), 36
registerNonPeriodic() (tamaas._tamaas.ModelFactory static method), 49
registerVolumeOperators() (tamaas._tamaas.ModelFactory static method), 49
RegularizedPowerlaw1D (class in tamaas._tamaas), 51
RegularizedPowerlaw2D (class in tamaas._tamaas), 52
relaxation (tamaas._tamaas.AndersonMixing property), 36
relaxation (tamaas._tamaas.EPICSolver property), 40
remotes (tamaas._tamaas.TamaasInfo attribute), 57
reset() (tamaas.nonlinear_solvers.DFSANESolver method), 63
reset() (tamaas.nonlinear_solvers.NewtonKrylovSolver method), 64
Residual (class in tamaas._tamaas), 52
rmsHeights() (tamaas._tamaas.Isopowerlaw1D method), 43
rmsHeights() (tamaas._tamaas.Isopowerlaw2D method), 44
rmsSlopes() (tamaas._tamaas.Isopowerlaw1D method), 43
rmsSlopes() (tamaas._tamaas.Isopowerlaw2D method), 44

S

SCALAR_OP (C macro), 178
SCALAR_OPERATOR (C macro), 177
SCALAR_OPERATOR_IMPL (C macro), 177
scatter() (in module tamaas._tamaas.mpi), 59
seeded_surfaces() (in module tamaas.utils), 65
sequential (class in tamaas._tamaas.mpi), 59
set_log_level() (in module tamaas._tamaas), 58
setDumpFrequency() (tamaas._tamaas.BeckTe-boulle method), 37
setDumpFrequency() (tamaas._tamaas.Condat method), 39

`setDumpFrequency()` (*tamaas._tamaas.ContactSolver method*), 39
`setDumpFrequency()` (*tamaas._tamaas.Kato method*), 44
`setDumpFrequency()` (*tamaas._tamaas.KatoSaturated method*), 45
`setDumpFrequency()` (*tamaas._tamaas.PolonskyKeerKey method*), 50
`setDumpFrequency()` (*tamaas._tamaas.PolonskyKeerTan method*), 51
`setElasticity()` (*tamaas._tamaas.Model method*), 48
`setFilter()` (*tamaas._tamaas.SurfaceGeneratorFilter1D method*), 55
`setFilter()` (*tamaas._tamaas.SurfaceGeneratorFilter2D method*), 56
`setFilter()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase1D method*), 56
`setFilter()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase2D method*), 57
`setIntegralOperator()` (*tamaas._tamaas.KatoSaturated method*), 45
`setIntegralOperator()` (*tamaas._tamaas.PolonskyKeerKey method*), 50
`setIntegrationMethod()` (*tamaas._tamaas.Model method*), 48
`setIntegrationMethod()` (*tamaas._tamaas.ModelFactory static method*), 49
`setIntegrationMethod()` (*tamaas._tamaas.Residual method*), 53
`setMaxIterations()` (*tamaas._tamaas.BeckTeBoulle method*), 37
`setMaxIterations()` (*tamaas._tamaas.Condat method*), 39
`setMaxIterations()` (*tamaas._tamaas.ContactSolver method*), 39
`setMaxIterations()` (*tamaas._tamaas.Kato method*), 44
`setMaxIterations()` (*tamaas._tamaas.KatoSaturated method*), 45
`setMaxIterations()` (*tamaas._tamaas.PolonskyKeerKey method*), 50
`setMaxIterations()` (*tamaas._tamaas.PolonskyKeerTan method*), 51
`setParameters()` (*tamaas._tamaas.AdhesionFunctional method*), 35
`setParameters()` (*tamaas._tamaas.ExponentialAdhesionFunctional method*), 41
`setParameters()` (*tamaas._tamaas.MaugisAdhesionFunctional method*), 47
`setParameters()` (*tamaas._tamaas.SquaredExponentialAdhesionFunctional method*), 53
`setRandomSeed()` (*tamaas._tamaas.SurfaceGenerator1D method*), 54
`setRandomSeed()` (*tamaas._tamaas.SurfaceGenerator2D method*), 54
`setRandomSeed()` (*tamaas._tamaas.SurfaceGeneratorFilter1D method*), 55
`setRandomSeed()` (*tamaas._tamaas.SurfaceGeneratorFilter2D method*), 56
`setRandomSeed()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase1D method*), 56
`setRandomSeed()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase2D method*), 57
`setSizes()` (*tamaas._tamaas.SurfaceGenerator1D method*), 54
`setSizes()` (*tamaas._tamaas.SurfaceGenerator2D method*), 54
`setSizes()` (*tamaas._tamaas.SurfaceGeneratorFilter1D method*), 55
`setSizes()` (*tamaas._tamaas.SurfaceGeneratorFilter2D method*), 56
`setSizes()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase1D method*), 56
`setSizes()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase2D method*), 57
`setSpectrum()` (*tamaas._tamaas.SurfaceGeneratorFilter1D method*), 55
`setSpectrum()` (*tamaas._tamaas.SurfaceGeneratorFilter2D method*), 56
`setSpectrum()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase1D method*), 56
`setSpectrum()` (*tamaas._tamaas.SurfaceGeneratorRandomPhase2D method*), 57
`setToleranceManager()` (*tamaas._tamaas.DFSANESolver method*), 59
`setToleranceManager()` (*tamaas._tamaas.EP-Solver method*), 40
`setToleranceManager()` (*tamaas.nonlinear_solvers.DFSANESolver method*), 63
`setToleranceManager()` (*tamaas.nonlinear_solvers.NewtonKrylovSolver method*), 64
`shape` (*tamaas._tamaas.IntegralOperator property*), 43
`shape` (*tamaas._tamaas.Model property*), 48
`shape` (*tamaas._tamaas.SurfaceGenerator1D property*), 54
`shape` (*tamaas._tamaas.SurfaceGenerator2D property*), 55
`shape` (*tamaas._tamaas.SurfaceGeneratorFilter1D property*), 55
`shape` (*tamaas._tamaas.SurfaceGeneratorFilter2D property*), 56
`shape` (*tamaas._tamaas.SurfaceGeneratorRandomPhase1D property*), 56
`shape` (*tamaas._tamaas.SurfaceGeneratorRandomPhase2D property*), 57

singleton (C++ member), 179
size() (in module `tamaas._tamaas.mpi`), 59
solve() (`tamaas._tamaas.DFSANESolver` method), 59
solve() (`tamaas._tamaas.AndersonMixing` method), 36
solve() (`tamaas._tamaas.BeckTeboulle` method), 37
solve() (`tamaas._tamaas.Condat` method), 39
solve() (`tamaas._tamaas.ContactSolver` method), 39
solve() (`tamaas._tamaas.EPICSolver` method), 40
solve() (`tamaas._tamaas.EPSolver` method), 40
solve() (`tamaas._tamaas.Kato` method), 44
solve() (`tamaas._tamaas.KatoSaturated` method), 45
solve() (`tamaas._tamaas.PolonskyKeerRey` method), 50
solve() (`tamaas._tamaas.PolonskyKeerTan` method), 51
solve() (`tamaas.nonlinear_solvers.DFSANESolver` method), 63
solve() (`tamaas.nonlinear_solvers.NewtonKrylovSolver` method), 64
solveDirichlet() (`tamaas._tamaas.BEEngine` method), 36
solveDirichlet() (`tamaas._tamaas.Model` method), 48
solveNeumann() (`tamaas._tamaas.BEEngine` method), 36
solveNeumann() (`tamaas._tamaas.Model` method), 48
solveRegularized() (`tamaas._tamaas.Kato` method), 44
solveRelaxed() (`tamaas._tamaas.Kato` method), 44
solveTresca() (`tamaas._tamaas.PolonskyKeerTan` method), 51
spectrum (`tamaas._tamaas.SurfaceGeneratorFilter1D` property), 55
spectrum (`tamaas._tamaas.SurfaceGeneratorFilter2D` property), 56
spectrum (`tamaas._tamaas.SurfaceGeneratorRandomPhase1D` property), 56
spectrum (`tamaas._tamaas.SurfaceGeneratorRandomPhase2D` property), 57
SquaredExponentialAdhesionFunctional (class in `tamaas._tamaas`), 53
Statistics1D (class in `tamaas._tamaas`), 53
Statistics2D (class in `tamaas._tamaas`), 53
surface (`tamaas._tamaas.BeckTeboulle` property), 37
surface (`tamaas._tamaas.Condat` property), 39
surface (`tamaas._tamaas.ContactSolver` property), 39
surface (`tamaas._tamaas.Kato` property), 45
surface (`tamaas._tamaas.KatoSaturated` property), 45
surface (`tamaas._tamaas.PolonskyKeerRey` property), 50
surface (`tamaas._tamaas.PolonskyKeerTan` property), 51
surface_1d (`tamaas._tamaas.model_type` attribute), 58
surface_2d (`tamaas._tamaas.model_type` attribute), 58
SurfaceGenerator1D (class in `tamaas._tamaas`), 54
SurfaceGenerator2D (class in `tamaas._tamaas`), 54

SurfaceGeneratorFilter1D (class in `tamaas._tamaas`), 55
SurfaceGeneratorFilter2D (class in `tamaas._tamaas`), 55
SurfaceGeneratorRandomPhase1D (class in `tamaas._tamaas`), 56
SurfaceGeneratorRandomPhase2D (class in `tamaas._tamaas`), 56
SWITCH_DISPATCH_CASE (C macro), 184
system_size (`tamaas._tamaas.Model` property), 48

T

tamaas
 module, 35
tamaas (C++ type), 166
tamaas._tamaas
 module, 35
tamaas._tamaas.compute
 module, 59
tamaas._tamaas.mpi
 module, 59
tamaas.dumpers
 module, 60
tamaas.nonlinear_solvers
 module, 63
tamaas.utils
 module, 64
tamaas::Accumulator (C++ class), 66
tamaas::Accumulator::acc_range (C++ struct), 65
tamaas::Accumulator::acc_range::_begin (C++ member), 66
tamaas::Accumulator::acc_range::_end (C++ member), 66
tamaas::Accumulator::acc_range::begin (C++ function), 65
tamaas::Accumulator::acc_range::end (C++ function), 65
tamaas::Accumulator::Accumulator (C++ function), 66
tamaas::Accumulator::accumulator (C++ member), 67
tamaas::Accumulator::backward (C++ function), 66
tamaas::Accumulator::BufferType (C++ type), 67
tamaas::Accumulator::direction (C++ enum), 66
tamaas::Accumulator::direction::_backward (C++ enumerator), 66
tamaas::Accumulator::direction::_forward (C++ enumerator), 66
tamaas::Accumulator::elements (C++ function), 66

tamaas::Accumulator::forward (C++ *function*), 66

tamaas::Accumulator::iterator (C++ *struct*), 114

tamaas::Accumulator::iterator::acc (C++ *member*), 115

tamaas::Accumulator::iterator::integ (C++ *type*), 114

tamaas::Accumulator::iterator::iterator (C++ *function*), 114

tamaas::Accumulator::iterator::k (C++ *member*), 115

tamaas::Accumulator::iterator::l (C++ *member*), 115

tamaas::Accumulator::iterator::layer (C++ *function*), 115

tamaas::Accumulator::iterator::next (C++ *function*), 115

tamaas::Accumulator::iterator::nextElement (C++ *function*), 115

tamaas::Accumulator::iterator::operator!= (C++ *function*), 114

tamaas::Accumulator::iterator::operator* (C++ *function*), 114

tamaas::Accumulator::iterator::operator++ (C++ *function*), 114

tamaas::Accumulator::iterator::r (C++ *member*), 115

tamaas::Accumulator::iterator::setup (C++ *function*), 115

tamaas::Accumulator::iterator::upper (C++ *member*), 114

tamaas::Accumulator::iterator::xc (C++ *member*), 115

tamaas::Accumulator::makeUniformMesh (C++ *function*), 66

tamaas::Accumulator::node_positions (C++ *member*), 67

tamaas::Accumulator::node_values (C++ *member*), 67

tamaas::Accumulator::nodePositions (C++ *function*), 66

tamaas::Accumulator::nodeValue (C++ *function*), 66

tamaas::Accumulator::reset (C++ *function*), 66

tamaas::Accumulator::trait (C++ *type*), 67

tamaas::Accumulator::waveVectors (C++ *function*), 66

tamaas::Accumulator::wavevectors (C++ *member*), 67

tamaas::allocateGrid (C++ *function*), 171

tamaas::Allocator (C++ *type*), 166

tamaas::AndersonMixing (C++ *class*), 67

tamaas::AndersonMixing::AndersonMixing (C++ *function*), 68

tamaas::AndersonMixing::computeGamma (C++ *function*), 68

tamaas::AndersonMixing::M (C++ *member*), 68

tamaas::AndersonMixing::memory_t (C++ *type*), 68

tamaas::AndersonMixing::mixingUpdate (C++ *function*), 68

tamaas::AndersonMixing::solve (C++ *function*), 68

tamaas::applyCompute (C++ *function*), 168

tamaas::Array (C++ *struct*), 69

tamaas::Array::~~Array (C++ *function*), 70

tamaas::Array::alloc_ (C++ *member*), 71

tamaas::Array::Array (C++ *function*), 70

tamaas::Array::data (C++ *function*), 70

tamaas::Array::operator= (C++ *function*), 70

tamaas::Array::operator[] (C++ *function*), 70

tamaas::Array::reserve (C++ *function*), 70

tamaas::Array::reserved_ (C++ *member*), 71

tamaas::Array::resize (C++ *function*), 70

tamaas::Array::size (C++ *function*), 70

tamaas::Array::view (C++ *function*), 71

tamaas::Array::view_ (C++ *member*), 71

tamaas::Array::wrap (C++ *function*), 70

tamaas::Array::wrapped_ (C++ *member*), 71

tamaas::assertion_error (C++ *class*), 71

tamaas::BeckTeboulle (C++ *class*), 71

tamaas::BeckTeboulle::BeckTeboulle (C++ *function*), 71

tamaas::BeckTeboulle::solve (C++ *function*), 71

tamaas::BeckTeboulle::solveTpl (C++ *function*), 71

tamaas::BEEngine (C++ *class*), 71

tamaas::BEEngine::~~BEEngine (C++ *function*), 72

tamaas::BEEngine::BEEngine (C++ *function*), 72

tamaas::BEEngine::getModel (C++ *function*), 72

tamaas::BEEngine::getNeumannNorm (C++ *function*), 72

tamaas::BEEngine::model (C++ *member*), 72

tamaas::BEEngine::operators (C++ *member*), 72

tamaas::BEEngine::registerDirichlet (C++ *function*), 72

tamaas::BEEngine::registerNeumann (C++ *function*), 72

tamaas::BEEngine::solveDirichlet (C++ *function*), 72

tamaas::BEEngine::solveNeumann (C++ *function*), 72

tamaas::BEEngineTpl (C++ *class*), 72

tamaas::BEEngineTpl::BEEngineTpl (C++ *function*), 72

tamaas::BEEngineTpl::registerDirichlet (C++ *function*), 72

tamaas::BEEngineTpl::registerNeumann (C++ *function*), 72

tamaas::BEEngineTpl::solveDirichlet (C++ *function*), 72

tamaas::BEEngineTpl::solveNeumann (C++ *function*), 72

tamaas::Boussinesq (C++ *class*), 73

tamaas::boussinesq (C++ *function*), 170

tamaas::Boussinesq::apply (C++ *function*), 73

tamaas::Boussinesq::Boussinesq (C++ *function*), 73

tamaas::Boussinesq::initialize (C++ *function*), 73

tamaas::Boussinesq::parent (C++ *type*), 73

tamaas::Boussinesq::trait (C++ *type*), 73

tamaas::cast_int (C++ *function*), 171

tamaas::checkLoopSize (C++ *function*), 168

tamaas::Cluster (C++ *class*), 76

tamaas::Cluster::BBox (C++ *type*), 77

tamaas::Cluster::boundingBox (C++ *function*), 76

tamaas::Cluster::Cluster (C++ *function*), 76

tamaas::Cluster::extent (C++ *function*), 76

tamaas::Cluster::getArea (C++ *function*), 76

tamaas::Cluster::getDiagonalNeighbors (C++ *function*), 76, 77

tamaas::Cluster::getNextNeighbors (C++ *function*), 76, 77

tamaas::Cluster::getPerimeter (C++ *function*), 76

tamaas::Cluster::getPoints (C++ *function*), 76

tamaas::Cluster::perimeter (C++ *member*), 77

tamaas::Cluster::Point (C++ *type*), 77

tamaas::Cluster::points (C++ *member*), 77

tamaas::Complex (C++ *type*), 166

tamaas::complex (C++ *type*), 166

tamaas::compute (C++ *type*), 172

tamaas::compute::Deviatoric (C++ *struct*), 83

tamaas::compute::Deviatoric::call (C++ *function*), 83

tamaas::compute::Eigenvalues (C++ *struct*), 84

tamaas::compute::Eigenvalues::call (C++ *function*), 84

tamaas::compute::VonMises (C++ *struct*), 162

tamaas::compute::VonMises::call (C++ *function*), 162

tamaas::Condat (C++ *class*), 78

tamaas::Condat::Condat (C++ *function*), 78

tamaas::Condat::pressure_old (C++ *member*), 78

tamaas::Condat::solve (C++ *function*), 78

tamaas::Condat::solveTpl (C++ *function*), 78

tamaas::Condat::updateGap (C++ *function*), 78

tamaas::Condat::updateLagrange (C++ *function*), 78

tamaas::ContactSolver (C++ *class*), 78

tamaas::ContactSolver::_gap (C++ *member*), 79

tamaas::ContactSolver::~~ContactSolver (C++ *function*), 78

tamaas::ContactSolver::addFunctionalTerm (C++ *function*), 79

tamaas::ContactSolver::ContactSolver (C++ *function*), 78

tamaas::ContactSolver::dump_frequency (C++ *member*), 79

tamaas::ContactSolver::functional (C++ *member*), 79

tamaas::ContactSolver::getDumpFrequency (C++ *function*), 79

tamaas::ContactSolver::getFunctional (C++ *function*), 79

tamaas::ContactSolver::getMaxIterations (C++ *function*), 78

tamaas::ContactSolver::getModel (C++ *function*), 79

tamaas::ContactSolver::getSurface (C++ *function*), 79

tamaas::ContactSolver::logIteration (C++ *function*), 78

tamaas::ContactSolver::max_iterations (C++ *member*), 79

tamaas::ContactSolver::model (C++ *member*), 79

tamaas::ContactSolver::printState (C++ *function*), 78

tamaas::ContactSolver::setDumpFrequency (C++ *function*), 79

tamaas::ContactSolver::setFunctional (C++ *function*), 79

tamaas::ContactSolver::setMaxIterations (C++ *function*), 78

tamaas::ContactSolver::solve (C++ *function*), 79

tamaas::ContactSolver::surface (C++ *member*), 79

tamaas::ContactSolver::surface_stddev (C++ *member*), 79

tamaas::ContactSolver::TAMAAS_ACCESSOR (C++ *function*), 79
 tamaas::ContactSolver::tolerance (C++ *member*), 79
 tamaas::createFromModelType (C++ *function*), 171
 tamaas::CuFFTEngine (C++ *class*), 79
 tamaas::CuFFTEngine::_flags (C++ *member*), 80
 tamaas::CuFFTEngine::backward (C++ *function*), 80
 tamaas::CuFFTEngine::backwardImpl (C++ *function*), 80
 tamaas::CuFFTEngine::cast (C++ *function*), 80
 tamaas::CuFFTEngine::CuFFTEngine (C++ *function*), 80
 tamaas::CuFFTEngine::flags (C++ *function*), 80
 tamaas::CuFFTEngine::forward (C++ *function*), 80
 tamaas::CuFFTEngine::forwardImpl (C++ *function*), 80
 tamaas::CuFFTEngine::getPlans (C++ *function*), 80
 tamaas::CuFFTEngine::plan_t (C++ *type*), 80
 tamaas::CuFFTEngine::plans (C++ *member*), 80
 tamaas::DCFFT (C++ *class*), 81
 tamaas::DCFFT::apply (C++ *function*), 81
 tamaas::DCFFT::bdim (C++ *member*), 82
 tamaas::DCFFT::comp (C++ *member*), 82
 tamaas::DCFFT::DCFFT (C++ *function*), 81
 tamaas::DCFFT::dim (C++ *member*), 82
 tamaas::DCFFT::extended_buffer (C++ *member*), 82
 tamaas::DCFFT::initInfluence (C++ *function*), 82
 tamaas::DCFFT::trait (C++ *type*), 81
 tamaas::dense (C++ *function*), 170
 tamaas::derivative_traits (C++ *struct*), 82
 tamaas::derivative_traits<0> (C++ *struct*), 82
 tamaas::derivative_traits<0>::out_components (C++ *member*), 82
 tamaas::derivative_traits<0>::source_components (C++ *member*), 82
 tamaas::derivative_traits<1> (C++ *struct*), 82
 tamaas::derivative_traits<1>::out_components (C++ *member*), 83
 tamaas::derivative_traits<1>::source_components (C++ *member*), 83
 tamaas::derivative_traits<2> (C++ *struct*), 83
 tamaas::derivative_traits<2>::out_components (C++ *member*), 83
 tamaas::derivative_traits<2>::source_components (C++ *member*), 83
 tamaas::detail (C++ *type*), 172
 tamaas::detail::append_to_stream (C++ *function*), 172
 tamaas::detail::Apply (C++ *struct*), 68
 tamaas::detail::Apply::apply (C++ *function*), 68
 tamaas::detail::Apply<0> (C++ *struct*), 68
 tamaas::detail::Apply<0>::apply (C++ *function*), 68
 tamaas::detail::ApplyFunctor (C++ *class*), 68
 tamaas::detail::ApplyFunctor::ApplyFunctor (C++ *function*), 69
 tamaas::detail::ApplyFunctor::functor (C++ *member*), 69
 tamaas::detail::ApplyFunctor::operator() (C++ *function*), 69
 tamaas::detail::areAllEqual (C++ *function*), 173
 tamaas::detail::boundary_fft_helper (C++ *struct*), 72
 tamaas::detail::boundary_fft_helper::backwardTransform (C++ *function*), 73
 tamaas::detail::boundary_fft_helper<m, m> (C++ *struct*), 73
 tamaas::detail::boundary_fft_helper<m, m>::backwardTransform (C++ *function*), 73
 tamaas::detail::BoussinesqHelper (C++ *struct*), 75
 tamaas::detail::BoussinesqHelper::accumulator (C++ *member*), 76
 tamaas::detail::BoussinesqHelper::apply (C++ *function*), 75
 tamaas::detail::BoussinesqHelper::bdim (C++ *member*), 76
 tamaas::detail::BoussinesqHelper::BufferType (C++ *type*), 75
 tamaas::detail::BoussinesqHelper::dim (C++ *member*), 76
 tamaas::detail::BoussinesqHelper::makeFundamentalModeGreatAgain (C++ *function*), 75
 tamaas::detail::BoussinesqHelper::out_t (C++ *type*), 75
 tamaas::detail::BoussinesqHelper::source_t (C++ *type*), 75

tamaas::detail::BoussinesqHelper::trait (C++ *type*), 75

tamaas::detail::ComputeOperator (C++ *class*), 77

tamaas::detail::ComputeOperator::apply (C++ *function*), 77

tamaas::detail::ComputeOperator::ComputeOperator (C++ *function*), 77

tamaas::detail::ComputeOperator::getKind (C++ *function*), 77

tamaas::detail::ComputeOperator::getType (C++ *function*), 77

tamaas::detail::ComputeOperator::updateFromModel (C++ *function*), 77

tamaas::detail::concat_args (C++ *function*), 172

tamaas::detail::dim_t (C++ *type*), 172

tamaas::detail::fold_trait_tail_rec (C++ *struct*), 94

tamaas::detail::get_rec<0, n, ns...> (C++ *struct*), 95

tamaas::detail::KelvinHelper (C++ *struct*), 122

tamaas::detail::KelvinHelper::~~KelvinHelper (C++ *function*), 123

tamaas::detail::KelvinHelper::accumulator (C++ *member*), 124

tamaas::detail::KelvinHelper::applyFreeTerm (C++ *function*), 123

tamaas::detail::KelvinHelper::applyIntegral (C++ *function*), 123

tamaas::detail::KelvinHelper::bdim (C++ *member*), 123

tamaas::detail::KelvinHelper::BufferType (C++ *type*), 123

tamaas::detail::KelvinHelper::cutoff_functor (C++ *struct*), 81

tamaas::detail::KelvinHelper::cutoff_functor::cutoff (C++ *member*), 81

tamaas::detail::KelvinHelper::cutoff_functor::dx (C++ *member*), 81

tamaas::detail::KelvinHelper::cutoff_functor::kelvin (C++ *member*), 81

tamaas::detail::KelvinHelper::cutoff_functor::operator() (C++ *function*), 81

tamaas::detail::KelvinHelper::cutoff_functor::r (C++ *member*), 81

tamaas::detail::KelvinHelper::cutoff_functor::xc (C++ *member*), 81

tamaas::detail::KelvinHelper::dim (C++ *member*), 123

tamaas::detail::KelvinHelper::makeFundamentalGreatAgain (C++ *function*), 123

tamaas::detail::KelvinHelper::out_t (C++ *type*), 123

tamaas::detail::KelvinHelper::source_t (C++ *type*), 123

tamaas::detail::KelvinHelper::trait (C++ *type*), 123

tamaas::detail::KelvinTrait (C++ *struct*), 124

tamaas::detail::KelvinTrait<influence::Boussinesq<dim, 0>> (C++ *struct*), 124

tamaas::detail::KelvinTrait<influence::Boussinesq<dim, 0>>::out_t (C++ *type*), 124

tamaas::detail::KelvinTrait<influence::Boussinesq<dim, 0>>::source_t (C++ *type*), 124

tamaas::detail::KelvinTrait<influence::Boussinesq<dim, 1>> (C++ *struct*), 124

tamaas::detail::KelvinTrait<influence::Boussinesq<dim, 1>>::out_t (C++ *type*), 124

tamaas::detail::KelvinTrait<influence::Boussinesq<dim, 1>>::source_t (C++ *type*), 124

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 0>> (C++ *struct*), 124

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 0>>::out_t (C++ *type*), 124

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 0>>::source_t (C++ *type*), 124

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 1>> (C++ *struct*), 124

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 1>>::out_t (C++ *type*), 125

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 1>>::source_t (C++ *type*), 125

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 2>> (C++ *struct*), 125

tamaas::detail::KelvinTrait<influence::Kelvin<dim, 2>>::out_t (C++ *type*), 125

tamaas::detail::KelvinTrait<influ-

ence::Kelvin<dim, 2>>::source_t (C++ type), 125
 tamaas::detail::loopSize (C++ function), 172
 tamaas::detail::model_type_t (C++ type), 172
 tamaas::detail::product_tail_rec (C++ struct), 142
 tamaas::detail::reduction_helper (C++ struct), 144
 tamaas::detail::reduction_helper<operation::max, ReturnType> (C++ struct), 144
 tamaas::detail::reduction_helper<operation::max, ReturnType>::init (C++ function), 144
 tamaas::detail::reduction_helper<operation::min, ReturnType> (C++ struct), 144
 tamaas::detail::reduction_helper<operation::min, ReturnType>::init (C++ function), 144
 tamaas::detail::reduction_helper<operation::plus, ReturnType> (C++ struct), 144
 tamaas::detail::reduction_helper<operation::plus, ReturnType>::init (C++ function), 144
 tamaas::detail::reduction_helper<operation::times, ReturnType> (C++ struct), 144
 tamaas::detail::reduction_helper<operation::times, ReturnType>::init (C++ function), 145
 tamaas::detail::static_switch_dispatch (C++ function), 173
 tamaas::detail::SurfaceTractionHelper (C++ struct), 156
 tamaas::detail::SurfaceTractionHelper::accumulator (C++ member), 157
 tamaas::detail::SurfaceTractionHelper::bdim (C++ member), 157
 tamaas::detail::SurfaceTractionHelper::BufferType (C++ type), 156
 tamaas::detail::SurfaceTractionHelper::computeSurfaceTractions (C++ function), 157
 tamaas::detail::SurfaceTractionHelper::dim (C++ member), 157
 tamaas::detail::SurfaceTractionHelper::kelvin_t (C++ type), 156
 tamaas::detail::SurfaceTractionHelper::out_t (C++ type), 156
 tamaas::detail::SurfaceTractionHelper::source_t (C++ type), 156
 tamaas::detail::SurfaceTractionHelper::trait (C++ type), 156
 tamaas::detail::tuple_dispatch (C++ function), 173
 tamaas::detail::tuple_dispatch_with_default (C++ function), 173
 tamaas::deviatoric (C++ function), 168
 tamaas::DFSANESolver (C++ class), 83
 tamaas::DFSANESolver::computeAlpha (C++ function), 84
 tamaas::DFSANESolver::computeSearchDirection (C++ function), 84
 tamaas::DFSANESolver::computeSpectralCoeff (C++ function), 84
 tamaas::DFSANESolver::current_x (C++ member), 84
 tamaas::DFSANESolver::delta_residual (C++ member), 84
 tamaas::DFSANESolver::delta_x (C++ member), 84
 tamaas::DFSANESolver::DFSANESolver (C++ function), 83
 tamaas::DFSANESolver::eta (C++ member), 84
 tamaas::DFSANESolver::lineSearch (C++ function), 84
 tamaas::DFSANESolver::max_iterations (C++ member), 84
 tamaas::DFSANESolver::previous_merits (C++ member), 84
 tamaas::DFSANESolver::previous_residual (C++ member), 84
 tamaas::DFSANESolver::search_direction (C++ member), 84
 tamaas::DFSANESolver::solve (C++ function), 83
 tamaas::DFSANESolver::TAMAAS_ACCESSOR (C++ function), 83
 tamaas::dimension_dispatch (C++ function), 171
 tamaas::dummy (C++ member), 172
 tamaas::eigenvalues (C++ function), 168, 170
 tamaas::EPICSolver (C++ class), 86
 tamaas::EPICSolver::acceleratedSolve (C++ function), 86
 tamaas::EPICSolver::computeError (C++ function), 86
 tamaas::EPICSolver::csolver (C++ member), 87
 tamaas::EPICSolver::EPICSolver (C++ function), 86
 tamaas::EPICSolver::epsolver (C++ member), 86

ber), 87

tamaas::EPICSolver::fixedPoint (C++ *function*), 86

tamaas::EPICSolver::getModel (C++ *function*), 86

tamaas::EPICSolver::max_iterations (C++ *member*), 87

tamaas::EPICSolver::pressure (C++ *member*), 87

tamaas::EPICSolver::pressure_inc (C++ *member*), 87

tamaas::EPICSolver::relaxation (C++ *member*), 87

tamaas::EPICSolver::residual_disp (C++ *member*), 87

tamaas::EPICSolver::setViews (C++ *function*), 86

tamaas::EPICSolver::solve (C++ *function*), 86

tamaas::EPICSolver::surface (C++ *member*), 87

tamaas::EPICSolver::TAMAAS_ACCESSOR (C++ *function*), 86

tamaas::EPICSolver::tolerance (C++ *member*), 87

tamaas::EPSolver (C++ *class*), 87

tamaas::EPSolver::_residual (C++ *member*), 88

tamaas::EPSolver::_x (C++ *member*), 88

tamaas::EPSolver::~~EPSolver (C++ *function*), 87

tamaas::EPSolver::abs_tol (C++ *member*), 88

tamaas::EPSolver::beforeSolve (C++ *function*), 87

tamaas::EPSolver::EPSolver (C++ *function*), 87

tamaas::EPSolver::getResidual (C++ *function*), 87

tamaas::EPSolver::getStrainIncrement (C++ *function*), 87

tamaas::EPSolver::getTolerance (C++ *function*), 87

tamaas::EPSolver::setTolerance (C++ *function*), 87

tamaas::EPSolver::setToleranceManager (C++ *function*), 87

tamaas::EPSolver::solve (C++ *function*), 87

tamaas::EPSolver::updateState (C++ *function*), 87

tamaas::Exception (C++ *class*), 88

tamaas::Exception::~~Exception (C++ *function*), 88

tamaas::Exception::Exception (C++ *function*), 88

tamaas::Exception::msg (C++ *member*), 88

tamaas::Exception::what (C++ *function*), 88

tamaas::exchange (C++ *function*), 170

tamaas::ExponentialElement (C++ *struct*), 88

tamaas::ExponentialElement<1> (C++ *struct*), 88

tamaas::ExponentialElement<1>::g0 (C++ *function*), 89

tamaas::ExponentialElement<1>::g1 (C++ *function*), 89

tamaas::ExponentialElement<1>::shapes (C++ *function*), 89

tamaas::ExponentialElement<1>::sign (C++ *function*), 89

tamaas::factorize (C++ *function*), 171

tamaas::FFTEngine (C++ *class*), 89

tamaas::FFTEngine::~~FFTEngine (C++ *function*), 89

tamaas::FFTEngine::backward (C++ *function*), 89

tamaas::FFTEngine::computeFrequencies (C++ *function*), 89

tamaas::FFTEngine::forward (C++ *function*), 89

tamaas::FFTEngine::key_t (C++ *type*), 90

tamaas::FFTEngine::make_key (C++ *function*), 90

tamaas::FFTEngine::makeEngine (C++ *function*), 89

tamaas::FFTEngine::realCoefficients (C++ *function*), 89

tamaas::FFTWAllocator (C++ *struct*), 90

tamaas::FFTWAllocator::allocate (C++ *function*), 90

tamaas::FFTWAllocator::deallocate (C++ *function*), 90

tamaas::FFTWEngine (C++ *class*), 90

tamaas::FFTWEngine::_flags (C++ *member*), 91

tamaas::FFTWEngine::backward (C++ *function*), 90

tamaas::FFTWEngine::backwardImpl (C++ *function*), 91

tamaas::FFTWEngine::cast (C++ *function*), 91

tamaas::FFTWEngine::complex_t (C++ *type*), 91

tamaas::FFTWEngine::FFTWEngine (C++ *function*), 90

tamaas::FFTWEngine::flags (C++ *function*), 90

tamaas::FFTWEngine::forward (C++ *function*), 90

tamaas::FFTWEngine::forwardImpl (C++ *function*), 91

tamaas::FFTWEngine::getPlans (C++ *function*), 91

tamaas::FFTWEngine::plan_t (C++ *type*), 91
 tamaas::FFTWEngine::plans (C++ *member*), 91
 tamaas::FFTWMPIDEngine (C++ *class*), 91
 tamaas::FFTWMPIDEngine::backward (C++ *function*), 91
 tamaas::FFTWMPIDEngine::FFTWEngine (C++ *function*), 92
 tamaas::FFTWMPIDEngine::forward (C++ *function*), 91
 tamaas::FFTWMPIDEngine::getPlans (C++ *function*), 92
 tamaas::FFTWMPIDEngine::local_size (C++ *function*), 92
 tamaas::FFTWMPIDEngine::make_key (C++ *function*), 92
 tamaas::FFTWMPIDEngine::workspaces (C++ *member*), 92
 tamaas::FieldContainer (C++ *struct*), 92
 tamaas::FieldContainer::~~FieldContainer (C++ *function*), 93
 tamaas::FieldContainer::at (C++ *function*), 93
 tamaas::FieldContainer::field (C++ *function*), 93
 tamaas::FieldContainer::fields (C++ *function*), 93
 tamaas::FieldContainer::fields_ (C++ *member*), 93
 tamaas::FieldContainer::fields_map (C++ *function*), 93
 tamaas::FieldContainer::FieldsMap (C++ *type*), 92
 tamaas::FieldContainer::GridBasePtr (C++ *type*), 92
 tamaas::FieldContainer::Key (C++ *type*), 92
 tamaas::FieldContainer::operator[] (C++ *function*), 93
 tamaas::FieldContainer::request (C++ *function*), 93
 tamaas::FieldContainer::Value (C++ *type*), 92
 tamaas::Filter (C++ *class*), 93
 tamaas::Filter::~~Filter (C++ *function*), 94
 tamaas::Filter::computeFilter (C++ *function*), 94
 tamaas::Filter::Filter (C++ *function*), 94
 tamaas::finalize (C++ *function*), 170
 tamaas::FloodFill (C++ *class*), 94
 tamaas::FloodFill::getClusters (C++ *function*), 94
 tamaas::FloodFill::getSegments (C++ *function*), 94
 tamaas::FloodFill::getVolumes (C++ *function*), 94
 tamaas::FloodFill::List (C++ *type*), 94
 tamaas::fold_trait (C++ *struct*), 94
 tamaas::functional (C++ *type*), 173
 tamaas::functional::AdhesionFunctional (C++ *class*), 67
 tamaas::functional::AdhesionFunctional::AdhesionFunctional (C++ *function*), 67
 tamaas::functional::AdhesionFunctional::getParameters (C++ *function*), 67
 tamaas::functional::AdhesionFunctional::parameters (C++ *member*), 67
 tamaas::functional::AdhesionFunctional::setParameters (C++ *function*), 67
 tamaas::functional::AdhesionFunctional::surface (C++ *member*), 67
 tamaas::functional::ElasticFunctional (C++ *class*), 84
 tamaas::functional::ElasticFunctional::buffer (C++ *member*), 85
 tamaas::functional::ElasticFunctional (C++ *function*), 85
 tamaas::functional::ElasticFunctional::op (C++ *member*), 85
 tamaas::functional::ElasticFunctional::surface (C++ *member*), 85
 tamaas::functional::ElasticFunctional-Gap (C++ *class*), 85
 tamaas::functional::ElasticFunctional-Gap::computeF (C++ *function*), 85
 tamaas::functional::ElasticFunctional-Gap::computeGradF (C++ *function*), 85
 tamaas::functional::ElasticFunctional-Gap::ElasticFunctional (C++ *function*), 85
 tamaas::functional::ElasticFunctional-Pressure (C++ *class*), 85
 tamaas::functional::ElasticFunctional-Pressure::computeF (C++ *function*), 85
 tamaas::functional::ElasticFunctional-Pressure::computeGradF (C++ *function*), 85
 tamaas::functional::ElasticFunctional-Pressure::ElasticFunctional (C++ *function*), 85
 tamaas::functional::ExponentialAdhesionFunctional (C++ *class*), 88
 tamaas::functional::ExponentialAdhe-

sionFunctional::computeF (C++
function), 88

tamaas::functional::ExponentialAdhe-
sionFunctional::computeGradF
(C++ function), 88

tamaas::functional::ExponentialAdhe-
sionFunctional::Exponential-
AdhesionFunctional (C++ function),
88

tamaas::functional::Functional (C++
class), 94

tamaas::functional::Functional::~~Func-
tional (C++ function), 95

tamaas::functional::Functional::com-
puteF (C++ function), 95

tamaas::functional::Functional::com-
puteGradF (C++ function), 95

tamaas::functional::MaugisAdhesion-
Functional (C++ class), 128

tamaas::functional::MaugisAdhesion-
Functional::computeF (C++ function),
128

tamaas::functional::MaugisAdhesion-
Functional::computeGradF (C++
function), 128

tamaas::functional::MaugisAdhesion-
Functional::MaugisAdhesionFunc-
tional (C++ function), 128

tamaas::functional::MetaFunctional (C++
class), 128

tamaas::functional::MetaFunc-
tional::addFunctionalTerm (C++
function), 128

tamaas::functional::MetaFunc-
tional::clear (C++ function), 128

tamaas::functional::MetaFunc-
tional::computeF (C++ function),
128

tamaas::functional::MetaFunc-
tional::computeGradF (C++ function),
128

tamaas::functional::MetaFunc-
tional::FunctionalList (C++ type),
128

tamaas::functional::MetaFunc-
tional::functionals (C++ member),
128

tamaas::functional::SquaredExponen-
tialAdhesionFunctional (C++ class),
148

tamaas::functional::SquaredExponen-
tialAdhesionFunctional::com-
puteF (C++ function), 148

tamaas::functional::SquaredExponen-
tialAdhesionFunctional::com-
puteGradF (C++ function), 148

tamaas::functional::SquaredExponen-
tialAdhesionFunctional::Square-
dExponentialAdhesionFunctional
(C++ function), 148

tamaas::get (C++ struct), 95

tamaas::Grid (C++ class), 95

tamaas::Grid::~~Grid (C++ function), 96

tamaas::Grid::computeOffset (C++ function),
98

tamaas::Grid::computeSize (C++ function), 96

tamaas::Grid::computeStrides (C++ func-
tion), 96

tamaas::Grid::copy (C++ function), 97

tamaas::Grid::dimension (C++ member), 97

tamaas::Grid::getDimension (C++ function),
96

tamaas::Grid::getStrides (C++ function), 97

tamaas::Grid::Grid (C++ function), 96

tamaas::Grid::init (C++ function), 98

tamaas::Grid::is_valid_index (C++ type), 98

tamaas::Grid::move (C++ function), 97

tamaas::Grid::n (C++ member), 97

tamaas::Grid::operator () (C++ function), 97

tamaas::Grid::operator= (C++ function), 97

tamaas::Grid::printself (C++ function), 96

tamaas::Grid::reference (C++ type), 95

tamaas::Grid::resize (C++ function), 96

tamaas::Grid::sizes (C++ function), 96

tamaas::Grid::strides (C++ member), 97

tamaas::Grid::unpackOffset (C++ function),
98

tamaas::Grid::value_type (C++ type), 95

tamaas::Grid::wrap (C++ function), 97

tamaas::GridBase (C++ class), 98

tamaas::GridBase::~~GridBase (C++ function),
99

tamaas::GridBase::begin (C++ function), 99

tamaas::GridBase::const_iterator (C++
type), 98

tamaas::GridBase::copy (C++ function), 100

tamaas::GridBase::data (C++ member), 101

tamaas::GridBase::dataSize (C++ function),
99

tamaas::GridBase::dot (C++ function), 100

tamaas::GridBase::end (C++ function), 99

tamaas::GridBase::getDimension (C++ func-
tion), 99

tamaas::GridBase::getGlobalNbPoints
(C++ function), 99

tamaas::GridBase::getInternalData (C++
function), 99

tamaas::GridBase::getNbComponents (C++ *function*), 99
 tamaas::GridBase::getNbPoints (C++ *function*), 99
 tamaas::GridBase::globalDataSize (C++ *function*), 99
 tamaas::GridBase::GridBase (C++ *function*), 98, 99
 tamaas::GridBase::iterator (C++ *type*), 98
 tamaas::GridBase::l2norm (C++ *function*), 100
 tamaas::GridBase::max (C++ *function*), 100
 tamaas::GridBase::mean (C++ *function*), 100
 tamaas::GridBase::min (C++ *function*), 100
 tamaas::GridBase::move (C++ *function*), 100
 tamaas::GridBase::nb_components (C++ *member*), 101
 tamaas::GridBase::operator() (C++ *function*), 99
 tamaas::GridBase::operator*= (C++ *function*), 100, 101
 tamaas::GridBase::operator+= (C++ *function*), 99--101
 tamaas::GridBase::operator/= (C++ *function*), 100, 101
 tamaas::GridBase::operator= (C++ *function*), 100, 101
 tamaas::GridBase::operator-= (C++ *function*), 100, 101
 tamaas::GridBase::reference (C++ *type*), 98
 tamaas::GridBase::reserve (C++ *function*), 99
 tamaas::GridBase::resize (C++ *function*), 99
 tamaas::GridBase::setNbComponents (C++ *function*), 99
 tamaas::GridBase::sum (C++ *function*), 100
 tamaas::GridBase::uniformSetComponents (C++ *function*), 99
 tamaas::GridBase::value_type (C++ *type*), 98
 tamaas::GridBase::var (C++ *function*), 100
 tamaas::GridBase::view (C++ *function*), 99
 tamaas::GridBase::wrap (C++ *function*), 101
 tamaas::GridHermitian (C++ *class*), 101
 tamaas::GridHermitian::GridHermitian (C++ *function*), 102
 tamaas::GridHermitian::hermitianDimensions (C++ *function*), 102
 tamaas::GridHermitian::operator() (C++ *function*), 102
 tamaas::GridHermitian::packTuple (C++ *function*), 102
 tamaas::GridHermitian::value_type (C++ *type*), 102
 tamaas::GridView (C++ *class*), 102
 tamaas::GridView::~~GridView (C++ *function*), 103
 tamaas::GridView::begin (C++ *function*), 103
 tamaas::GridView::const_iterator (C++ *type*), 102
 tamaas::GridView::dataSize (C++ *function*), 103
 tamaas::GridView::end (C++ *function*), 103
 tamaas::GridView::grid (C++ *member*), 103
 tamaas::GridView::GridView (C++ *function*), 103
 tamaas::GridView::iterator (C++ *type*), 102
 tamaas::GridView::reference (C++ *type*), 102
 tamaas::GridView::reserve (C++ *function*), 103
 tamaas::GridView::resize (C++ *function*), 103
 tamaas::GridView::value_type (C++ *type*), 102
 tamaas::Hooke (C++ *class*), 104
 tamaas::Hooke::apply (C++ *function*), 105
 tamaas::Hooke::getKind (C++ *function*), 105
 tamaas::Hooke::getType (C++ *function*), 105
 tamaas::Hooke::IntegralOperator (C++ *function*), 105
 tamaas::Hooke::matvec (C++ *function*), 105
 tamaas::Hooke::matvecShape (C++ *function*), 105
 tamaas::Hooke::updateFromModel (C++ *function*), 105
 tamaas::HookeField (C++ *class*), 105
 tamaas::HookeField::apply (C++ *function*), 105
 tamaas::HookeField::HookeField (C++ *function*), 105
 tamaas::HookeFieldHelper (C++ *struct*), 105
 tamaas::HookeFieldHelper::elasticity (C++ *member*), 105
 tamaas::HookeFieldHelper::operator() (C++ *function*), 105
 tamaas::influence (C++ *type*), 173
 tamaas::influence::Boussinesq (C++ *class*), 73
 tamaas::influence::Boussinesq<3, 0> (C++ *class*), 73
 tamaas::influence::Boussinesq<3, 0>::applyU0 (C++ *function*), 74
 tamaas::influence::Boussinesq<3, 0>::applyU1 (C++ *function*), 74
 tamaas::influence::Boussinesq<3, 0>::Boussinesq (C++ *function*), 74
 tamaas::influence::Boussinesq<3, 0>::dim (C++ *member*), 74
 tamaas::influence::Boussinesq<3, 0>::lambda (C++ *member*), 74
 tamaas::influence::Boussinesq<3,

`0>::mu (C++ member), 74`
`tamaas::influence::Boussinesq<3,`
`0>::nu (C++ member), 74`
`tamaas::influence::Boussinesq<3,`
`0>::order (C++ member), 74`
`tamaas::influence::Boussinesq<3, 1>`
`(C++ class), 74`
`tamaas::influence::Boussinesq<3,`
`1>::applyU0 (C++ function), 74`
`tamaas::influence::Boussinesq<3,`
`1>::applyU1 (C++ function), 74`
`tamaas::influence::Boussinesq<3,`
`1>::dim (C++ member), 75`
`tamaas::influence::Boussinesq<3,`
`1>::order (C++ member), 75`
`tamaas::influence::Boussinesq<3,`
`1>::parent (C++ type), 75`
`tamaas::influence::computeD (C++ function),`
`173`
`tamaas::influence::computeD2 (C++ func-`
`tion), 173`
`tamaas::influence::ElasticHelper (C++`
`struct), 85`
`tamaas::influence::ElasticHelper::ElasticHelper (C++`
`function), 86`
`tamaas::influence::ElasticHelper::in-`
`verse (C++ function), 86`
`tamaas::influence::ElasticHelper::lambda (C++ member),`
`86`
`tamaas::influence::ElasticHelper::mu`
`(C++ member), 86`
`tamaas::influence::ElasticHelper::nu`
`(C++ member), 86`
`tamaas::influence::ElasticHelper::op-`
`erator() (C++ function), 86`
`tamaas::influence::Kelvin (C++ class), 119`
`tamaas::influence::Kelvin<3, 0> (C++`
`class), 120`
`tamaas::influence::Kelvin<3, 0>::ap-`
`plyU0 (C++ function), 121`
`tamaas::influence::Kelvin<3, 0>::ap-`
`plyU1 (C++ function), 121`
`tamaas::influence::Kelvin<3, 0>::b (C++`
`member), 121`
`tamaas::influence::Kelvin<3, 0>::dim`
`(C++ member), 121`
`tamaas::influence::Kelvin<3,`
`0>::Kelvin (C++ function), 121`
`tamaas::influence::Kelvin<3, 0>::mu`
`(C++ member), 121`
`tamaas::influence::Kelvin<3, 0>::order`
`(C++ member), 121`
`tamaas::influence::Kelvin<3, 1> (C++`
`class), 121`
`tamaas::influence::Kelvin<3, 1>::ap-`
`plyU0 (C++ function), 121`
`tamaas::influence::Kelvin<3, 1>::ap-`
`plyU1 (C++ function), 121`
`tamaas::influence::Kelvin<3, 1>::dim`
`(C++ member), 122`
`tamaas::influence::Kelvin<3, 1>::order`
`(C++ member), 122`
`tamaas::influence::Kelvin<3, 1>::par-`
`ent (C++ type), 122`
`tamaas::influence::Kelvin<3, 2> (C++`
`class), 122`
`tamaas::influence::Kelvin<3, 2>::ap-`
`plyDiscontinuityTerm (C++ function),`
`122`
`tamaas::influence::Kelvin<3, 2>::ap-`
`plyU0 (C++ function), 122`
`tamaas::influence::Kelvin<3, 2>::ap-`
`plyU1 (C++ function), 122`
`tamaas::influence::Kelvin<3, 2>::dim`
`(C++ member), 122`
`tamaas::influence::Kelvin<3, 2>::order`
`(C++ member), 122`
`tamaas::influence::Kelvin<3, 2>::par-`
`ent (C++ type), 122`
`tamaas::influence::[anonymous] (C++ type),`
`173`
`tamaas::initialize (C++ function), 170`
`tamaas::Int (C++ type), 166`
`tamaas::IntegralOperator (C++ class), 105`
`tamaas::IntegralOperator::apply (C++`
`function), 106`
`tamaas::IntegralOperator::applyIf (C++`
`function), 106`
`tamaas::IntegralOperator::getKind (C++`
`function), 106`
`tamaas::IntegralOperator::getModel (C++`
`function), 106`
`tamaas::IntegralOperator::getOpera-`
`torNorm (C++ function), 106`
`tamaas::IntegralOperator::getType (C++`
`function), 106`
`tamaas::IntegralOperator::IntegralOp-`
`erator (C++ function), 106`
`tamaas::IntegralOperator::kind (C++`
`enum), 106`
`tamaas::IntegralOperator::kind::dirac`
`(C++ enumerator), 106`
`tamaas::IntegralOperator::kind::dirichlet (C++ enu-`
`merator), 106`
`tamaas::IntegralOperator::kind::neu-`

mann (C++ *enumerator*), 106
 tamaas::IntegralOperator::matvec (C++ *function*), 106
 tamaas::IntegralOperator::matvecShape (C++ *function*), 106
 tamaas::IntegralOperator::model (C++ *member*), 107
 tamaas::IntegralOperator::updateFromModel (C++ *function*), 106
 tamaas::integration_method (C++ *enum*), 167
 tamaas::integration_method::cutoff (C++ *enumerator*), 167
 tamaas::integration_method::linear (C++ *enumerator*), 167
 tamaas::Integrator (C++ *class*), 107
 tamaas::Integrator::bounds (C++ *member*), 107
 tamaas::Integrator::element (C++ *type*), 107
 tamaas::Integrator::F (C++ *function*), 107
 tamaas::Integrator::G0 (C++ *function*), 107
 tamaas::Integrator::G1 (C++ *function*), 107
 tamaas::Internal (C++ *struct*), 107
 tamaas::Internal::field (C++ *member*), 108
 tamaas::Internal::initialize (C++ *function*), 107
 tamaas::Internal::operator std::shared_ptr<GridBase<T>> (C++ *function*), 108
 tamaas::Internal::operator= (C++ *function*), 107
 tamaas::Internal::previous (C++ *member*), 108
 tamaas::Internal::reset (C++ *function*), 107
 tamaas::Internal::update (C++ *function*), 107
 tamaas::invariants (C++ *function*), 170
 tamaas::is_arithmetic (C++ *struct*), 108
 tamaas::is_arithmetic<thrust::complex<T>> (C++ *struct*), 108
 tamaas::is_policy (C++ *struct*), 108
 tamaas::is_policy<const thrust::detail::device_t> (C++ *struct*), 108
 tamaas::is_policy<const thrust::detail::device_t&> (C++ *struct*), 108
 tamaas::is_policy<const thrust::detail::host_t> (C++ *struct*), 108
 tamaas::is_policy<const thrust::detail::host_t&> (C++ *struct*), 108
 tamaas::is_policy<thrust::detail::device_t> (C++ *struct*), 108
 tamaas::is_policy<thrust::detail::host_t> (C++ *struct*), 109
 tamaas::is_proxy (C++ *struct*), 109
 tamaas::is_proxy<MatrixProxy<T, n, m>> (C++ *struct*), 109
 tamaas::is_proxy<SymMatrixProxy<T, n>> (C++ *struct*), 109
 tamaas::is_proxy<TensorProxy<StaticParent, T, dims...>> (C++ *struct*), 109
 tamaas::is_proxy<VectorProxy<T, n>> (C++ *struct*), 109
 tamaas::Isopowerlaw (C++ *class*), 109
 tamaas::Isopowerlaw::alpha (C++ *function*), 110
 tamaas::Isopowerlaw::computeFilter (C++ *function*), 109
 tamaas::Isopowerlaw::elasticEnergy (C++ *function*), 110
 tamaas::Isopowerlaw::hurst (C++ *member*), 110
 tamaas::Isopowerlaw::moments (C++ *function*), 109
 tamaas::Isopowerlaw::operator() (C++ *function*), 109
 tamaas::Isopowerlaw::q0 (C++ *member*), 110
 tamaas::Isopowerlaw::q1 (C++ *member*), 110
 tamaas::Isopowerlaw::q2 (C++ *member*), 110
 tamaas::Isopowerlaw::radialPSDMoment (C++ *function*), 110
 tamaas::Isopowerlaw::rmsHeights (C++ *function*), 109
 tamaas::Isopowerlaw::rmsSlopes (C++ *function*), 110
 tamaas::Isopowerlaw::TAMAAS_ACCESSOR (C++ *function*), 110
 tamaas::IsotropicHardening (C++ *class*), 110
 tamaas::IsotropicHardening::applyTangent (C++ *function*), 111
 tamaas::IsotropicHardening::CMat (C++ *type*), 111
 tamaas::IsotropicHardening::computeEigenStress (C++ *function*), 110
 tamaas::IsotropicHardening::computeInelasticDeformationIncrement (C++ *function*), 110
 tamaas::IsotropicHardening::computeStress (C++ *function*), 110
 tamaas::IsotropicHardening::cumulated_plastic_strain (C++ *member*), 111
 tamaas::IsotropicHardening::dim (C++ *member*), 112
 tamaas::IsotropicHardening::Field (C++ *type*), 111
 tamaas::IsotropicHardening::getHardeningModulus (C++ *function*), 111
 tamaas::IsotropicHardening::getPlasticStrain (C++ *function*), 111

tamaas::IsotropicHardening::getYieldStress (C++ *function*), 111

tamaas::IsotropicHardening::h (C++ *member*), 111

tamaas::IsotropicHardening::hardening (C++ *function*), 111

tamaas::IsotropicHardening::IsotropicHardening (C++ *function*), 110

tamaas::IsotropicHardening::Mat (C++ *type*), 111

tamaas::IsotropicHardening::parent (C++ *type*), 111

tamaas::IsotropicHardening::plastic_strain (C++ *member*), 111

tamaas::IsotropicHardening::setHardeningModulus (C++ *function*), 111

tamaas::IsotropicHardening::setYieldStress (C++ *function*), 111

tamaas::IsotropicHardening::sigma_0 (C++ *member*), 111

tamaas::IsotropicHardening::trait (C++ *type*), 111

tamaas::IsotropicHardening::type (C++ *member*), 112

tamaas::IsotropicHardening::update (C++ *function*), 111

tamaas::iterator_ (C++ *type*), 173

tamaas::iterator_::iterator (C++ *class*), 112

tamaas::iterator_::iterator::data (C++ *member*), 113

tamaas::iterator_::iterator::difference_type (C++ *type*), 112

tamaas::iterator_::iterator::iterator (C++ *function*), 112

tamaas::iterator_::iterator::iterator_category (C++ *type*), 112

tamaas::iterator_::iterator::operator!= (C++ *function*), 113

tamaas::iterator_::iterator::operator* (C++ *function*), 112

tamaas::iterator_::iterator::operator++ (C++ *function*), 113

tamaas::iterator_::iterator::operator+= (C++ *function*), 113

tamaas::iterator_::iterator::operator< (C++ *function*), 113

tamaas::iterator_::iterator::operator= (C++ *function*), 112

tamaas::iterator_::iterator::operator== (C++ *function*), 113

tamaas::iterator_::iterator::operator- (C++ *function*), 113

tamaas::iterator_::iterator::pointer (C++ *type*), 112

tamaas::iterator_::iterator::reference (C++ *type*), 112

tamaas::iterator_::iterator::setStep (C++ *function*), 113

tamaas::iterator_::iterator::step (C++ *member*), 113

tamaas::iterator_::iterator::value_type (C++ *type*), 112

tamaas::iterator_::iterator_view (C++ *class*), 115

tamaas::iterator_::iterator_view::computeAccessOffset (C++ *function*), 116

tamaas::iterator_::iterator_view::difference_type (C++ *type*), 115

tamaas::iterator_::iterator_view::iterator_view (C++ *function*), 116

tamaas::iterator_::iterator_view::n (C++ *member*), 116

tamaas::iterator_::iterator_view::operator= (C++ *function*), 116

tamaas::iterator_::iterator_view::pointer (C++ *type*), 115

tamaas::iterator_::iterator_view::reference (C++ *type*), 115

tamaas::iterator_::iterator_view::strides (C++ *member*), 116

tamaas::iterator_::iterator_view::tuple (C++ *member*), 116

tamaas::iterator_::iterator_view::value_type (C++ *type*), 115

tamaas::Kato (C++ *class*), 116

tamaas::Kato::addUniform (C++ *function*), 117

tamaas::Kato::computeCost (C++ *function*), 116

tamaas::Kato::computeFinalGap (C++ *function*), 117

tamaas::Kato::computeGradient (C++ *function*), 117

tamaas::Kato::computeMean (C++ *function*), 117

tamaas::Kato::computeValuesForCost (C++ *function*), 117

tamaas::Kato::computeValuesForCost-Tresca (C++ *function*), 117

tamaas::Kato::enforcePressureConstraints (C++ *function*), 117

tamaas::Kato::enforcePressureCoulomb (C++ *function*), 117

tamaas::Kato::enforcePressureMean (C++ *function*), 117

tamaas::Kato::enforcePressureTresca

(C++ function), 117

tamaas::Kato::engine (C++ member), 118

tamaas::Kato::gap (C++ member), 118

tamaas::Kato::initSurfaceWithComponents (C++ function), 117

tamaas::Kato::Kato (C++ function), 116

tamaas::Kato::mu (C++ member), 118

tamaas::Kato::N (C++ member), 118

tamaas::Kato::pressure (C++ member), 118

tamaas::Kato::regularize (C++ function), 118

tamaas::Kato::solve (C++ function), 116

tamaas::Kato::solveRegularized (C++ function), 116

tamaas::Kato::solveRegularizedTpl (C++ function), 117

tamaas::Kato::solveRelaxed (C++ function), 116

tamaas::Kato::solveRelaxedTpl (C++ function), 116

tamaas::Kato::solveTpl (C++ function), 116

tamaas::Kato::surfaceComp (C++ member), 118

tamaas::KatoSaturated (C++ class), 118

tamaas::KatoSaturated::~~KatoSaturated (C++ function), 118

tamaas::KatoSaturated::computeCriticalStep (C++ function), 118

tamaas::KatoSaturated::computeError (C++ function), 118

tamaas::KatoSaturated::computeSquaredNorm (C++ function), 118

tamaas::KatoSaturated::enforceAdmissibleState (C++ function), 119

tamaas::KatoSaturated::enforceMeanValue (C++ function), 118

tamaas::KatoSaturated::getPMax (C++ function), 119

tamaas::KatoSaturated::KatoSaturated (C++ function), 118

tamaas::KatoSaturated::meanOnUnsaturated (C++ function), 118

tamaas::KatoSaturated::pmax (C++ member), 119

tamaas::KatoSaturated::setPMax (C++ function), 119

tamaas::KatoSaturated::solve (C++ function), 118

tamaas::KatoSaturated::updatePrimal (C++ function), 118

tamaas::KatoSaturated::updateSearchDirection (C++ function), 118

tamaas::Kelvin (C++ class), 119

tamaas::Kelvin::applyIf (C++ function), 119

tamaas::Kelvin::cutoff (C++ member), 120

tamaas::Kelvin::cutoffIntegral (C++ function), 120

tamaas::Kelvin::dtrait (C++ type), 120

tamaas::Kelvin::filter_t (C++ type), 120

tamaas::Kelvin::Kelvin (C++ function), 119

tamaas::Kelvin::KelvinInfluence (C++ type), 120

tamaas::Kelvin::ktrait (C++ type), 120

tamaas::Kelvin::linearIntegral (C++ function), 120

tamaas::Kelvin::matvec (C++ function), 119

tamaas::Kelvin::matvecShape (C++ function), 119

tamaas::Kelvin::method (C++ member), 120

tamaas::Kelvin::Out (C++ type), 120

tamaas::Kelvin::parent (C++ type), 120

tamaas::Kelvin::setIntegrationMethod (C++ function), 119

tamaas::Kelvin::Source (C++ type), 120

tamaas::Kelvin::trait (C++ type), 120

tamaas::Logger (C++ class), 125

tamaas::Logger::~~Logger (C++ function), 125

tamaas::Logger::current_level (C++ member), 126

tamaas::Logger::get (C++ function), 125

tamaas::Logger::getCurrentLevel (C++ function), 125

tamaas::Logger::getWishLevel (C++ function), 125

tamaas::Logger::setLevel (C++ function), 125

tamaas::Logger::stream (C++ member), 126

tamaas::Logger::wish_level (C++ member), 126

tamaas::LogLevel (C++ enum), 167

tamaas::LogLevel::debug (C++ enumerator), 167

tamaas::LogLevel::error (C++ enumerator), 167

tamaas::LogLevel::info (C++ enumerator), 167

tamaas::LogLevel::warning (C++ enumerator), 167

tamaas::Loop (C++ class), 126

tamaas::Loop::arange (C++ class), 69

tamaas::Loop::arange::arange (C++ function), 69

tamaas::Loop::arange::begin (C++ function), 69

tamaas::Loop::arange::end (C++ function), 69

tamaas::Loop::arange::getNbComponents (C++ function), 69

tamaas::Loop::arange::it_type (C++ type), 69

tamaas::Loop::arange::range_size (C++ member), 69

tamaas::Loop::arange::reference (C++ type), 69

tamaas::Loop::arange::start (C++ member), 69

tamaas::Loop::Loop (C++ function), 126

tamaas::Loop::loop (C++ function), 126

tamaas::Loop::loopImpl (C++ function), 127

tamaas::Loop::range (C++ function), 126

tamaas::Loop::reduce (C++ function), 126

tamaas::Loop::reduceImpl (C++ function), 127

tamaas::Loop::reference_type (C++ type), 127

tamaas::Loop::transform_iterator (C++ class), 159

tamaas::Loop::transform_iterator::dereference (C++ function), 160

tamaas::Loop::transform_iterator::func (C++ member), 160

tamaas::Loop::transform_iterator::super_t (C++ type), 160

tamaas::Loop::transform_iterator::transform_iterator (C++ function), 160

tamaas::LUSubstitute (C++ function), 171

tamaas::make_component_view (C++ function), 168

tamaas::make_view (C++ function), 168

tamaas::Material (C++ class), 127

tamaas::Material::~~Material (C++ function), 127

tamaas::Material::applyTangent (C++ function), 127

tamaas::Material::computeEigenStress (C++ function), 127

tamaas::Material::computeStress (C++ function), 127

tamaas::Material::Field (C++ type), 127

tamaas::Material::Material (C++ function), 127

tamaas::Material::model (C++ member), 128

tamaas::Material::update (C++ function), 127

tamaas::Matrix (C++ type), 166

tamaas::MatrixProxy (C++ type), 166

tamaas::MFrontMaterial (C++ class), 128

tamaas::MFrontMaterial::dim (C++ member), 129

tamaas::MFrontMaterial::Field (C++ type), 129

tamaas::MFrontMaterial::MFrontMaterial (C++ function), 129

tamaas::MFrontMaterial::parent (C++ type), 129

tamaas::MFrontMaterial::trait (C++ type), 129

tamaas::MFrontMaterial::type (C++ member), 129

tamaas::Mindlin (C++ class), 129

tamaas::Mindlin::applyIf (C++ function), 129

tamaas::Mindlin::cutoffIntegral (C++ function), 129

tamaas::Mindlin::filter_t (C++ type), 130

tamaas::Mindlin::linearIntegral (C++ function), 129

tamaas::Mindlin::Mindlin (C++ function), 129

tamaas::Mindlin::parent (C++ type), 130

tamaas::Mindlin::surface_tractions (C++ member), 130

tamaas::Mindlin::trait (C++ type), 130

tamaas::Model (C++ class), 130

tamaas::Model::addDumper (C++ function), 132

tamaas::Model::applyElasticity (C++ function), 131

tamaas::Model::discretization (C++ member), 132

tamaas::Model::dump (C++ function), 132

tamaas::Model::dumpers (C++ member), 132

tamaas::Model::E (C++ member), 132

tamaas::Model::engine (C++ member), 132

tamaas::Model::getBEEngine (C++ function), 131

tamaas::Model::getBoundaryDiscretization (C++ function), 131

tamaas::Model::getBoundaryFields (C++ function), 131

tamaas::Model::getBoundarySystemSize (C++ function), 130

tamaas::Model::getDiscretization (C++ function), 130

tamaas::Model::getDisplacement (C++ function), 131

tamaas::Model::getGlobalDiscretization (C++ function), 131

tamaas::Model::getHertzModulus (C++ function), 130

tamaas::Model::getIntegralOperator (C++ function), 131

tamaas::Model::getIntegralOperators (C++ function), 131

tamaas::Model::getIntegralOperatorsMap (C++ function), 131

tamaas::Model::getPoissonRatio (C++ function), 130

tamaas::Model::getShearModulus (C++ function), 130

tamaas::Model::getSystemSize (C++ function), 130

tamaas::Model::getTraction (C++ function), 130

131

tamaas::Model::getType (C++ *function*), 130

tamaas::Model::getYoungModulus (C++ *function*), 130

tamaas::Model::isBoundaryField (C++ *function*), 131

tamaas::Model::Model (C++ *function*), 132

tamaas::Model::nu (C++ *member*), 132

tamaas::Model::operator<< (C++ *function*), 132

tamaas::Model::operators (C++ *member*), 132

tamaas::Model::registerIntegralOperator (C++ *function*), 131

tamaas::Model::setElasticity (C++ *function*), 130

tamaas::Model::setIntegrationMethod (C++ *function*), 131

tamaas::Model::setPoissonRatio (C++ *function*), 130

tamaas::Model::setYoungModulus (C++ *function*), 130

tamaas::Model::solveDirichlet (C++ *function*), 131

tamaas::Model::solveNeumann (C++ *function*), 131

tamaas::Model::system_size (C++ *member*), 132

tamaas::Model::updateOperators (C++ *function*), 131

tamaas::model_type (C++ *enum*), 167

tamaas::model_type::basic_1d (C++ *enumerator*), 167

tamaas::model_type::basic_2d (C++ *enumerator*), 168

tamaas::model_type::surface_1d (C++ *enumerator*), 168

tamaas::model_type::surface_2d (C++ *enumerator*), 168

tamaas::model_type::volume_1d (C++ *enumerator*), 168

tamaas::model_type::volume_2d (C++ *enumerator*), 168

tamaas::model_type_dispatch (C++ *function*), 171

tamaas::model_type_error (C++ *class*), 132

tamaas::model_type_traits (C++ *struct*), 132

tamaas::model_type_traits<model_type::basic_1d> (C++ *struct*), 132

tamaas::model_type_traits<model_type::basic_1d>::boundary_dimension (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_1d>::components (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_1d>::dimension (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_1d>::indices (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_1d>::repr (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_1d>::voigt (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_2d> (C++ *struct*), 133

tamaas::model_type_traits<model_type::basic_2d>::boundary_dimension (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_2d>::components (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_2d>::dimension (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_2d>::indices (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_2d>::repr (C++ *member*), 133

tamaas::model_type_traits<model_type::basic_2d>::voigt (C++ *member*), 133

tamaas::model_type_traits<model_type::surface_1d> (C++ *struct*), 133

tamaas::model_type_traits<model_type::surface_1d>::boundary_dimension (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_1d>::components (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_1d>::dimension (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_1d>::indices (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_1d>::repr (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_1d>::voigt (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_2d> (C++ *struct*), 134

tamaas::model_type_traits<model_type::surface_2d>::boundary_dimension (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_2d>::components (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_2d>::dimension (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_2d>::indices (C++ *member*), 134

tamaas::model_type_traits<model_type::surface_2d>::repr (C++ *member*), 134
tamaas::model_type_traits<model_type::surface_2d>::voigt (C++ *member*), 134
tamaas::model_type_traits<model_type::volume_1d> (C++ *struct*), 134
tamaas::model_type_traits<model_type::volume_1d>::boundary_dimension (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_1d>::components (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_1d>::dimension (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_1d>::indices (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_1d>::repr (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_1d>::voigt (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_2d> (C++ *struct*), 135
tamaas::model_type_traits<model_type::volume_2d>::boundary_dimension (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_2d>::components (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_2d>::dimension (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_2d>::indices (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_2d>::repr (C++ *member*), 135
tamaas::model_type_traits<model_type::volume_2d>::voigt (C++ *member*), 135
tamaas::ModelDumper (C++ *class*), 135
tamaas::ModelDumper::~ModelDumper (C++ *function*), 135
tamaas::ModelDumper::dump (C++ *function*), 135
tamaas::ModelFactory (C++ *class*), 135
tamaas::ModelFactory::createModel (C++ *function*), 136
tamaas::ModelFactory::createResidual (C++ *function*), 136
tamaas::ModelFactory::registerNonPeriodic (C++ *function*), 136
tamaas::ModelFactory::registerVolumeOperators (C++ *function*), 136
tamaas::ModelFactory::setIntegrationMethod (C++ *function*), 136
tamaas::ModelTemplate (C++ *class*), 136
tamaas::ModelTemplate::dim (C++ *member*), 137
tamaas::ModelTemplate::displacement_view (C++ *member*), 137
tamaas::ModelTemplate::getBoundaryDiscretization (C++ *function*), 136
tamaas::ModelTemplate::getBoundarySystemSize (C++ *function*), 136
tamaas::ModelTemplate::getGlobalDiscretization (C++ *function*), 136
tamaas::ModelTemplate::getType (C++ *function*), 136
tamaas::ModelTemplate::initializeBEEngine (C++ *function*), 137
tamaas::ModelTemplate::ModelTemplate (C++ *function*), 136
tamaas::ModelTemplate::setIntegrationMethod (C++ *function*), 136
tamaas::ModelTemplate::traction_view (C++ *member*), 137
tamaas::ModelTemplate::trait (C++ *type*), 137
tamaas::ModelTemplate::ViewType (C++ *type*), 137
tamaas::mpi_dummy (C++ *type*), 173
tamaas::mpi_dummy::abort (C++ *function*), 174
tamaas::mpi_dummy::allreduce (C++ *function*), 174
tamaas::mpi_dummy::bcast (C++ *function*), 174
tamaas::mpi_dummy::comm (C++ *struct*), 77
tamaas::mpi_dummy::comm::world (C++ *member*), 77
tamaas::mpi_dummy::finalize (C++ *function*), 174
tamaas::mpi_dummy::finalized (C++ *function*), 174
tamaas::mpi_dummy::gather (C++ *function*), 174
tamaas::mpi_dummy::init (C++ *function*), 174
tamaas::mpi_dummy::init_thread (C++ *function*), 174
tamaas::mpi_dummy::initialized (C++ *function*), 174
tamaas::mpi_dummy::rank (C++ *function*), 174
tamaas::mpi_dummy::reduce (C++ *function*), 174
tamaas::mpi_dummy::scatter (C++ *function*), 174
tamaas::mpi_dummy::scatterv (C++ *function*), 174
tamaas::mpi_dummy::sequential (C++ *struct*), 147
tamaas::mpi_dummy::sequential::enter (C++ *function*), 147
tamaas::mpi_dummy::sequential::exit

(C++ function), 147
 tamaas::mpi_dummy::sequential_guard (C++ struct), 147
 tamaas::mpi_dummy::sequential_guard::~~sequential_guard (C++ function), 147
 tamaas::mpi_dummy::sequential_guard::sequential_guard (C++ function), 147
 tamaas::mpi_dummy::size (C++ function), 174
 tamaas::mpi_dummy::thread (C++ enum), 174
 tamaas::mpi_dummy::thread::funneled (C++ enumerator), 174
 tamaas::mpi_dummy::thread::multiple (C++ enumerator), 174
 tamaas::mpi_dummy::thread::serialized (C++ enumerator), 174
 tamaas::mpi_dummy::thread::single (C++ enumerator), 174
 tamaas::nan_error (C++ class), 137
 tamaas::no_convergence_error (C++ struct), 137
 tamaas::no_convergence_error::error (C++ member), 137
 tamaas::no_convergence_error::no_convergence_error (C++ function), 137
 tamaas::no_convergence_error::tolerance (C++ member), 137
 tamaas::no_convergence_error::what (C++ function), 137
 tamaas::not_implemented_error (C++ class), 137
 tamaas::operation (C++ enum), 167
 tamaas::operation::max (C++ enumerator), 167
 tamaas::operation::min (C++ enumerator), 167
 tamaas::operation::plus (C++ enumerator), 167
 tamaas::operation::times (C++ enumerator), 167
 tamaas::operator* (C++ function), 169, 170
 tamaas::operator+ (C++ function), 169
 tamaas::operator<< (C++ function), 168, 171
 tamaas::operator- (C++ function), 169
 tamaas::outer (C++ function), 170
 tamaas::Partitioner (C++ struct), 137
 tamaas::Partitioner::alloc_size (C++ function), 138
 tamaas::Partitioner::cast_size (C++ function), 138
 tamaas::Partitioner::gather (C++ function), 138
 tamaas::Partitioner::global_size (C++ function), 138
 tamaas::Partitioner::local_offset (C++ function), 138
 tamaas::Partitioner::local_size (C++ function), 138
 tamaas::Partitioner::scatter (C++ function), 138
 tamaas::PolonskyKeerRey (C++ class), 139
 tamaas::PolonskyKeerRey::~~PolonskyKeerRey (C++ function), 140
 tamaas::PolonskyKeerRey::computeCriticalStep (C++ function), 140
 tamaas::PolonskyKeerRey::computeError (C++ function), 140
 tamaas::PolonskyKeerRey::computeSquaredNorm (C++ function), 140
 tamaas::PolonskyKeerRey::constraint_type (C++ member), 141
 tamaas::PolonskyKeerRey::defaultOperator (C++ function), 141
 tamaas::PolonskyKeerRey::displacement_view (C++ member), 141
 tamaas::PolonskyKeerRey::dual (C++ member), 141
 tamaas::PolonskyKeerRey::enforceAdmissibleState (C++ function), 140
 tamaas::PolonskyKeerRey::enforceMeanValue (C++ function), 140
 tamaas::PolonskyKeerRey::gap_view (C++ member), 141
 tamaas::PolonskyKeerRey::integral_op (C++ member), 141
 tamaas::PolonskyKeerRey::meanOnUnsaturated (C++ function), 140
 tamaas::PolonskyKeerRey::operation_type (C++ member), 141
 tamaas::PolonskyKeerRey::PolonskyKeerRey (C++ function), 140
 tamaas::PolonskyKeerRey::pressure_view (C++ member), 141
 tamaas::PolonskyKeerRey::primal (C++ member), 141
 tamaas::PolonskyKeerRey::projected_search_direction (C++ member), 141
 tamaas::PolonskyKeerRey::search_direction (C++ member), 141
 tamaas::PolonskyKeerRey::setIntegralOperator (C++ function), 140
 tamaas::PolonskyKeerRey::setupFunctional (C++ function), 140
 tamaas::PolonskyKeerRey::setViews (C++ function), 141
 tamaas::PolonskyKeerRey::solve (C++ function), 140
 tamaas::PolonskyKeerRey::type (C++ enum), 138

139

tamaas::PolonskyKeerRey::type::gap (C++ *enumerator*), 139

tamaas::PolonskyKeerRey::type::pres-
sure (C++ *enumerator*), 139

tamaas::PolonskyKeerRey::updatePrimal
(C++ *function*), 140

tamaas::PolonskyKeerRey::update-
SearchDirection (C++ *function*), 140

tamaas::PolonskyKeerRey::variable_type
(C++ *member*), 141

tamaas::PolonskyKeerTan (C++ *class*), 141

tamaas::PolonskyKeerTan::computeMean
(C++ *function*), 142

tamaas::PolonskyKeerTan::com-
puteSquaredNorm (C++ *function*), 142

tamaas::PolonskyKeerTan::computeStep-
Size (C++ *function*), 142

tamaas::PolonskyKeerTan::enforcePres-
sureMean (C++ *function*), 142

tamaas::PolonskyKeerTan::PolonskyKeer-
Tan (C++ *function*), 142

tamaas::PolonskyKeerTan::pro-
jected_search_direction (C++
member), 142

tamaas::PolonskyKeerTan::search_direc-
tion (C++ *member*), 142

tamaas::PolonskyKeerTan::search_direc-
tion_backup (C++ *member*), 142

tamaas::PolonskyKeerTan::solve (C++ *func-
tion*), 142

tamaas::PolonskyKeerTan::solveTpl (C++
function), 142

tamaas::PolonskyKeerTan::solveTresca
(C++ *function*), 142

tamaas::PolonskyKeerTan::truncate-
SearchDirection (C++ *function*), 142

tamaas::product (C++ *struct*), 142

tamaas::random_engine (C++ *type*), 167

tamaas::Range (C++ *class*), 143

tamaas::range (C++ *function*), 168

tamaas::Range::_begin (C++ *member*), 144

tamaas::Range::_end (C++ *member*), 144

tamaas::Range::begin (C++ *function*), 143

tamaas::Range::end (C++ *function*), 143

tamaas::Range::headless (C++ *function*), 143

tamaas::Range::is_valid_container (C++
struct), 109

tamaas::Range::iterator (C++ *class*), 113

tamaas::Range::iterator::iterator (C++
function), 114

tamaas::Range::iterator::operator* (C++
function), 114

tamaas::Range::iterator::parent (C++
type), 114

tamaas::Range::iterator::reference (C++
type), 113

tamaas::Range::iterator::value_type
(C++ *type*), 113

tamaas::Range::Range (C++ *function*), 143

tamaas::Range::reference (C++ *type*), 143

tamaas::Range::value_type (C++ *type*), 143

tamaas::Real (C++ *type*), 166

tamaas::registerWestergaardOperator
(C++ *function*), 170

tamaas::RegularizedPowerlaw (C++ *class*), 145

tamaas::RegularizedPowerlaw::compute-
Filter (C++ *function*), 145

tamaas::RegularizedPowerlaw::hurst (C++
member), 145

tamaas::RegularizedPowerlaw::opera-
tor() (C++ *function*), 145

tamaas::RegularizedPowerlaw::q1 (C++
member), 145

tamaas::RegularizedPowerlaw::q2 (C++
member), 145

tamaas::RegularizedPower-
law::TAMAAS_ACCESSOR (C++ *function*),
145

tamaas::Residual (C++ *class*), 145

tamaas::Residual::~~Residual (C++ *func-
tion*), 145

tamaas::Residual::applyTangent (C++ *func-
tion*), 145

tamaas::Residual::computeResidual (C++
function), 145

tamaas::Residual::computeResidualDis-
placement (C++ *function*), 145

tamaas::Residual::dim (C++ *member*), 147

tamaas::Residual::getMaterial (C++ *func-
tion*), 146

tamaas::Residual::getModel (C++ *func-
tion*), 146

tamaas::Residual::getVector (C++ *func-
tion*), 146

tamaas::Residual::integralOperator (C++
function), 146

tamaas::Residual::material (C++ *member*),
146

tamaas::Residual::model (C++ *member*), 146

tamaas::Residual::plastic_filter (C++
member), 146

tamaas::Residual::plastic_layers (C++
member), 146

tamaas::Residual::Residual (C++ *func-
tion*), 145

tamaas::Residual::residual (C++ *member*),
146

tamaas::Residual::strain (C++ member), 146
 tamaas::Residual::stress (C++ member), 146
 tamaas::Residual::tmp (C++ member), 146
 tamaas::Residual::type (C++ member), 147
 tamaas::Residual::updateFilter (C++ function), 146
 tamaas::Residual::updateState (C++ function), 145
 tamaas::Residual::voigt (C++ member), 147
 tamaas::solve (C++ function), 170
 tamaas::span (C++ struct), 147
 tamaas::span::begin (C++ function), 148
 tamaas::span::const_pointer (C++ type), 147
 tamaas::span::const_reference (C++ type), 148
 tamaas::span::data (C++ function), 148
 tamaas::span::data_ (C++ member), 148
 tamaas::span::difference_type (C++ type), 147
 tamaas::span::element_type (C++ type), 147
 tamaas::span::end (C++ function), 148
 tamaas::span::iterator (C++ type), 148
 tamaas::span::operator[] (C++ function), 148
 tamaas::span::pointer (C++ type), 147
 tamaas::span::reference (C++ type), 147
 tamaas::span::reverse_iterator (C++ type), 148
 tamaas::span::size (C++ function), 148
 tamaas::span::size_ (C++ member), 148
 tamaas::span::size_type (C++ type), 147
 tamaas::span::value_type (C++ type), 147
 tamaas::static_size_helper (C++ struct), 148
 tamaas::static_size_helper<StaticSymMatrix, n> (C++ struct), 149
 tamaas::StaticArray (C++ class), 149
 tamaas::StaticArray::_mem (C++ member), 151
 tamaas::StaticArray::~StaticArray (C++ function), 149
 tamaas::StaticArray::back (C++ function), 150
 tamaas::StaticArray::begin (C++ function), 150
 tamaas::StaticArray::copy (C++ function), 150
 tamaas::StaticArray::dot (C++ function), 149
 tamaas::StaticArray::end (C++ function), 150
 tamaas::StaticArray::front (C++ function), 150
 tamaas::StaticArray::l2norm (C++ function), 149
 tamaas::StaticArray::l2squared (C++ function), 149
 tamaas::StaticArray::operator() (C++ function), 149
 tamaas::StaticArray::operator= (C++ function), 149, 150
 tamaas::StaticArray::size (C++ member), 151
 tamaas::StaticArray::StaticArray (C++ function), 149
 tamaas::StaticArray::sum (C++ function), 149
 tamaas::StaticArray::T (C++ type), 151
 tamaas::StaticArray::T_bare (C++ type), 151
 tamaas::StaticArray::valid_size_t (C++ type), 151
 tamaas::StaticArray::value_type (C++ type), 149
 tamaas::StaticMatrix (C++ class), 151
 tamaas::StaticMatrix::deviatoric (C++ function), 151
 tamaas::StaticMatrix::fromSymmetric (C++ function), 151
 tamaas::StaticMatrix::mul (C++ function), 151
 tamaas::StaticMatrix::outer (C++ function), 151
 tamaas::StaticMatrix::T (C++ type), 152
 tamaas::StaticMatrix::T_bare (C++ type), 152
 tamaas::StaticMatrix::trace (C++ function), 151
 tamaas::StaticSymMatrix (C++ class), 152
 tamaas::StaticSymMatrix::deviatoric (C++ function), 152
 tamaas::StaticSymMatrix::operator+= (C++ function), 152
 tamaas::StaticSymMatrix::parent (C++ type), 152
 tamaas::StaticSymMatrix::sym_binary (C++ function), 152
 tamaas::StaticSymMatrix::symmetrize (C++ function), 152
 tamaas::StaticSymMatrix::T (C++ type), 152
 tamaas::StaticSymMatrix::trace (C++ function), 152
 tamaas::StaticTensor (C++ class), 152
 tamaas::StaticTensor::dim (C++ member), 153
 tamaas::StaticTensor::operator() (C++ function), 153
 tamaas::StaticTensor::parent (C++ type), 153
 tamaas::StaticTensor::T (C++ type), 153
 tamaas::StaticTensor::unpackOffset (C++ function), 153
 tamaas::StaticVector (C++ class), 153

tamaas::StaticVector::mul (C++ *function*), 153

tamaas::StaticVector::T (C++ *type*), 153

tamaas::Statistics (C++ *struct*), 153

tamaas::Statistics::computeAutocorrelation (C++ *function*), 154

tamaas::Statistics::computeFDRMSslope (C++ *function*), 154

tamaas::Statistics::computeMoments (C++ *function*), 154

tamaas::Statistics::computePowerSpectrum (C++ *function*), 154

tamaas::Statistics::computeRMSheights (C++ *function*), 154

tamaas::Statistics::computeSpectralRMSslope (C++ *function*), 154

tamaas::Statistics::contact (C++ *function*), 154

tamaas::Statistics::graphArea (C++ *function*), 154

tamaas::Statistics::PVector (C++ *type*), 154

tamaas::Statistics::rmsSlopesFromPSD (C++ *function*), 154

tamaas::SurfaceGenerator (C++ *class*), 154

tamaas::SurfaceGenerator::~~SurfaceGenerator (C++ *function*), 155

tamaas::SurfaceGenerator::buildSurface (C++ *function*), 155

tamaas::SurfaceGenerator::getRandomSeed (C++ *function*), 155

tamaas::SurfaceGenerator::getSizes (C++ *function*), 155

tamaas::SurfaceGenerator::global_size (C++ *member*), 155

tamaas::SurfaceGenerator::grid (C++ *member*), 155

tamaas::SurfaceGenerator::random_seed (C++ *member*), 155

tamaas::SurfaceGenerator::setRandomSeed (C++ *function*), 155

tamaas::SurfaceGenerator::setSize (C++ *function*), 155

tamaas::SurfaceGenerator::SurfaceGenerator (C++ *function*), 155

tamaas::SurfaceGeneratorFilter (C++ *class*), 155

tamaas::SurfaceGeneratorFilter::applyFilterOnSource (C++ *function*), 156

tamaas::SurfaceGeneratorFilter::buildSurface (C++ *function*), 155

tamaas::SurfaceGeneratorFilter::engine (C++ *member*), 156

tamaas::SurfaceGeneratorFilter::filter (C++ *member*), 156

tamaas::SurfaceGeneratorFilter::filter_coefficients (C++ *member*), 156

tamaas::SurfaceGeneratorFilter::generateWhiteNoise (C++ *function*), 156

tamaas::SurfaceGeneratorFilter::getSpectrum (C++ *function*), 155

tamaas::SurfaceGeneratorFilter::setFilter (C++ *function*), 155

tamaas::SurfaceGeneratorFilter::setSpectrum (C++ *function*), 155

tamaas::SurfaceGeneratorFilter::white_noise (C++ *member*), 156

tamaas::SurfaceGeneratorRandomPhase (C++ *class*), 156

tamaas::SurfaceGeneratorRandomPhase::buildSurface (C++ *function*), 156

tamaas::SymHookeFieldHelper (C++ *struct*), 157

tamaas::SymHookeFieldHelper::elasticity (C++ *member*), 157

tamaas::SymHookeFieldHelper::operator() (C++ *function*), 157

tamaas::SymMatrix (C++ *type*), 166

tamaas::SymMatrixProxy (C++ *type*), 166

tamaas::symmetrize (C++ *function*), 170

tamaas::TamaasInfo (C++ *struct*), 157

tamaas::TamaasInfo::backend (C++ *member*), 158

tamaas::TamaasInfo::branch (C++ *member*), 157

tamaas::TamaasInfo::build_type (C++ *member*), 157

tamaas::TamaasInfo::commit (C++ *member*), 157

tamaas::TamaasInfo::diff (C++ *member*), 158

tamaas::TamaasInfo::has_mpi (C++ *member*), 158

tamaas::TamaasInfo::remotes (C++ *member*), 157

tamaas::TamaasInfo::version (C++ *member*), 157

tamaas::Tensor (C++ *class*), 158

tamaas::Tensor::operator= (C++ *function*), 158

tamaas::Tensor::parent (C++ *type*), 158

tamaas::Tensor::size (C++ *member*), 158

tamaas::Tensor::Tensor (C++ *function*), 158

tamaas::TensorProxy (C++ *class*), 158

tamaas::TensorProxy::operator= (C++ *function*), 159

tamaas::TensorProxy::parent (C++ *type*), 159

tamaas::TensorProxy::stack_type (C++ type), 159

tamaas::TensorProxy::TensorProxy (C++ function), 159

tamaas::ToleranceManager (C++ struct), 159

tamaas::ToleranceManager::end_tol (C++ member), 159

tamaas::ToleranceManager::rate (C++ member), 159

tamaas::ToleranceManager::reset (C++ function), 159

tamaas::ToleranceManager::start_tol (C++ member), 159

tamaas::ToleranceManager::step (C++ function), 159

tamaas::ToleranceManager::tolerance (C++ member), 159

tamaas::ToleranceManager::ToleranceManager (C++ function), 159

tamaas::UInt (C++ type), 166

tamaas::UnifiedAllocator (C++ struct), 160

tamaas::UnifiedAllocator::allocate (C++ function), 160

tamaas::UnifiedAllocator::deallocate (C++ function), 160

tamaas::Vector (C++ type), 166

tamaas::VectorProxy (C++ type), 166

tamaas::voigt_size (C++ struct), 160

tamaas::voigt_size<1> (C++ struct), 160

tamaas::voigt_size<2> (C++ struct), 161

tamaas::voigt_size<3> (C++ struct), 161

tamaas::VolumePotential (C++ class), 161

tamaas::VolumePotential::apply (C++ function), 161

tamaas::VolumePotential::BufferType (C++ type), 161

tamaas::VolumePotential::engine (C++ member), 162

tamaas::VolumePotential::filter_t (C++ type), 161

tamaas::VolumePotential::getKind (C++ function), 161

tamaas::VolumePotential::getType (C++ function), 161

tamaas::VolumePotential::initialize (C++ function), 162

tamaas::VolumePotential::out_buffer (C++ member), 162

tamaas::VolumePotential::source_buffer (C++ member), 162

tamaas::VolumePotential::trait (C++ type), 162

tamaas::VolumePotential::transformOutput (C++ function), 162

tamaas::VolumePotential::transformSource (C++ function), 162

tamaas::VolumePotential::updateFromModel (C++ function), 161

tamaas::VolumePotential::VolumePotential (C++ function), 161

tamaas::VolumePotential::wavevectors (C++ member), 162

tamaas::vonMises (C++ function), 168

tamaas::Westergaard (C++ class), 162

tamaas::Westergaard::apply (C++ function), 163

tamaas::Westergaard::bdim (C++ member), 164

tamaas::Westergaard::buffer (C++ member), 163

tamaas::Westergaard::comp (C++ member), 164

tamaas::Westergaard::dim (C++ member), 164

tamaas::Westergaard::engine (C++ member), 163

tamaas::Westergaard::fourierApply (C++ function), 163

tamaas::Westergaard::getInfluence (C++ function), 163

tamaas::Westergaard::getKind (C++ function), 163

tamaas::Westergaard::getOperatorNorm (C++ function), 163

tamaas::Westergaard::getType (C++ function), 163

tamaas::Westergaard::influence (C++ member), 163

tamaas::Westergaard::initFromFunctor (C++ function), 163

tamaas::Westergaard::initInfluence (C++ function), 163

tamaas::Westergaard::matvec (C++ function), 163

tamaas::Westergaard::matvecShape (C++ function), 163

tamaas::Westergaard::trait (C++ type), 164

tamaas::Westergaard::updateFromModel (C++ function), 163

tamaas::Westergaard::Westergaard (C++ function), 163

tamaas::wrap_pbc (C++ function), 171

tamaas::[anonymous] (C++ type), 172

TAMAAS_ACCESSOR (C macro), 179

TAMAAS_ASSERT (C macro), 175

TAMAAS_FFTW_BACKEND (C macro), 179

TAMAAS_FFTW_BACKEND_NONE (C macro), 179

TAMAAS_FFTW_BACKEND_OMP (C macro), 179

TAMAAS_FFTW_BACKEND_THREADS (C macro), 179

TAMAAS_INT_TYPE (*C macro*), 180
TAMAAS_LOCATION (*C macro*), 175
TAMAAS_LOOP_BACKEND (*C macro*), 179
TAMAAS_LOOP_BACKEND_CPP (*C macro*), 179
TAMAAS_LOOP_BACKEND_CUDA (*C macro*), 179
TAMAAS_LOOP_BACKEND_OMP (*C macro*), 179
TAMAAS_LOOP_BACKEND_TBB (*C macro*), 179
TAMAAS_MODEL_TYPES (*C macro*), 184
TAMAAS_MSG (*C macro*), 175
TAMAAS_REAL_TYPE (*C macro*), 180
TAMAAS_USE_FFTW (*C macro*), 179
TamaasInfo (*class in tamaas._tamaas*), 57
THRUST_DEVICE_SYSTEM (*C macro*), 179
to_voigt() (*in module tamaas._tamaas*), 58
to_voigt() (*in module tamaas._tamaas.compute*), 60
tolerance (*tamaas._tamaas.DFSANESolver property*), 59
tolerance (*tamaas._tamaas.AndersonMixing property*), 36
tolerance (*tamaas._tamaas.BeckTeboulle property*), 37
tolerance (*tamaas._tamaas.Condat property*), 39
tolerance (*tamaas._tamaas.ContactSolver property*), 39
tolerance (*tamaas._tamaas.EPICSolver property*), 40
tolerance (*tamaas._tamaas.EPSolver property*), 40
tolerance (*tamaas._tamaas.Kato property*), 45
tolerance (*tamaas._tamaas.KatoSaturated property*), 45
tolerance (*tamaas._tamaas.PolonskyKeerRey property*), 50
tolerance (*tamaas._tamaas.PolonskyKeerTan property*), 51
tolerance (*tamaas.nonlinear_solvers.DFSANESolver property*), 63
tolerance (*tamaas.nonlinear_solvers.NewtonKrylovSolver property*), 64
ToleranceManager() (*in module tamaas.nonlinear_solvers*), 64
traction (*tamaas._tamaas.Model property*), 48
type (*tamaas._tamaas.IntegralOperator property*), 43
type (*tamaas._tamaas.Model property*), 48

U

updateFromModel() (*tamaas._tamaas.IntegralOperator method*), 43
updateState() (*tamaas._tamaas.DFSANESolver method*), 59
updateState() (*tamaas._tamaas.EPSolver method*), 40
updateState() (*tamaas._tamaas.Residual method*), 53
updateState() (*tamaas.nonlinear_solvers.DFSANESolver method*), 63
updateState() (*tamaas.nonlinear_solvers.NewtonKrylovSolver method*), 64

UVWDumper (*class in tamaas.dumpers*), 62
UVWGroupDumper (*class in tamaas.dumpers*), 62

V

value (*tamaas._tamaas.integration_method property*), 58
value (*tamaas._tamaas.KatoSaturated.type property*), 46
value (*tamaas._tamaas.LogLevel property*), 46
value (*tamaas._tamaas.model_type property*), 58
value (*tamaas._tamaas.PolonskyKeerRey.type property*), 51
VEC_OPERATOR (*C macro*), 177
VEC_OPERATOR_IMPL (*C macro*), 177
vector (*tamaas._tamaas.Residual property*), 53
VECTOR_OP (*C macro*), 178
version (*tamaas._tamaas.TamaasInfo attribute*), 57
volume_1d (*tamaas._tamaas.model_type attribute*), 58
volume_2d (*tamaas._tamaas.model_type attribute*), 58
von_mises() (*in module tamaas._tamaas.compute*), 60

W

warning (*tamaas._tamaas.LogLevel attribute*), 46
WESTERGAARD (*C macro*), 186
with_traceback() (*tamaas._tamaas.AssertionError method*), 36
with_traceback() (*tamaas._tamaas.FloatingPointError method*), 42
with_traceback() (*tamaas._tamaas.ModelTypeError method*), 49
with_traceback() (*tamaas._tamaas.NotImplementedError method*), 50
with_traceback() (*tamaas.nonlinear_solvers.NLNoConvergence method*), 63