nanite Documentation

Release 0.9.1

Paul Müller

Nov 16, 2018

Contents

Nanite is a Python library for loading, fitting, and rating AFM force-indentation data. This is the documentation of nanite version 0.9.1.

CHAPTER 1

Getting started

1.1 Installation

To install nanite, use one of the following methods (the package dependencies will be installed automatically):

- **from PyPI:** `pip install nanite[CLI]`
- **from sources:** `pip install -e .[CLI]`

The appendix `[CLI]` makes sure that all dependencies for the *command line interface* are installed. If you are only using nanite as a Python module, you may safely omit it.

Note that if you are installing from source or if no binary wheel is available for your platform and Python version, Cython will be installed to build the required nanite extensions. If this process fails, please request a binary wheel for your platform (e.g. Windows 64bit) and Python version (e.g. 3.6) by creating a new issue.

1.2 What is nanite?

The development of nanite was motivated by a unique problem that arises in AFM force-indentation data analysis, particularly for biological samples: The data quality varies a lot due to biological variation and due to experimental complexities that have to be dealt with when measuring biological samples. To address this problem, nanite makes use of machine-learning (á la scikit-learn), which allows to automatically determine the quality of a force-indentation curve based on a user-defined rating scheme (see *Rating workflow* for more information). But nanite is much more than just that. It comes with an extensive set of tools for AFM force-indentation data analysis.

1.3 Use cases

If you are a frequent AFM user, you might have run into several problems involving data analysis, ranging from simple data fitting to the visualization of quantitative force-indentation maps. Here are a few usage examples of nanite:

- You would like to automate your data analysis pipeline from loading force-indentation data to displaying a fit to the approach part with a Hertz model for a spherical indenter. You can do so with nanite, either via scripting or via the command-line interface that comes with nanite. For more information, see *Fitting guide*.
- You would like to automatically analyze and visualize maps of force-indentation data. This is possible with the *nanite.QMap* class.
- You would like to sort force-indentation data according to data quality using your own training set (not the one shipped with nanite). Nanite allows you to create your own training set from your own experimental data, locally. Besides that, you can make use of multiple regressors and visualize the rating e.g. of force-indentation maps. For an overview, see *Rating workflow*.

1.4 Basic usage

If you are not interested in scripting, please have a look at the *fitting guide*.

In a Python script, you may use nanite as follows:

```
In [1]: import nanite

In [2]: group = nanite.load_group("data/force-save-example.jpk-force")

In [3]: idnt = group[0]  # This group actually as only one indentation curve.

In [4]: idnt.apply_preprocessing(["compute_tip_position",
   ...:                           "correct_force_offset",
   ...:                           "correct_tip_offset"])
   ...:

In [5]: idnt.fit_model(model_key="sneddon_spher")

In [6]: idnt.rate_quality()  # 0 means bad, 10 means good quality
Out[6]: 9.03167147631572
```

You can find more examples in the *examples* section.

CHAPTER 2

Command-line interface

The nanite command-line interface (CLI) simplifies several functionalities of nanite, making fitting, rating, and the generation of training sets accessible to the user.

2.1 nanite-setup-profile

Set up a profile for fitting and rating. The profile is stored in the user's default configuration directory. Setting up a profile is required prior to running *nanite-fit* and *nanite-rate*.

```
usage: nanite-setup-profile [-h]
```

2.2 nanite-fit

Fit AFM force-indentation data. Statistics (.tsv file) and visualizations of the fits (multi-page .tif file) are stored in the results directory.

```
usage: nanite-fit [-h] data_path out_dir
```

positional arguments	
data_path	input folder containing AFM force-indentation data
out_dir	results directory

2.3 nanite-rate

Manually rate (the fit to) AFM force-indentation data. A graphical user interface allows to rate and comment on each data set. The fits and all data are stored in a rating container that can then be passed to *nanite-generate-training-set*.

```
usage: nanite-rate [-h] data_path rating_path
```

positional arguments	
data_path	input folder containing AFM force-indentation data
rating_path	path to the output rating container (will be created if it does not already exist)

2.4 nanite-generate-training-set

Create a training set for usage in nanite from rating containers (.h5 files manually created with *nanite-rate*).

```
usage: nanite-generate-training-set [-h] data_path out_dir
```

positional arguments	
data_path	path to a rating container or a folder containing rating containers
out_dir	directory where the training set will be stored

CHAPTER 3

Fitting guide

This is a summary of the methods used by nanite for fitting force-indentation data. Examples are given below.

3.1 Preprocessors

Prior to data analysis, force-indentation data has to be preprocessed. One of the most important preprocessing steps is to perform a tip-sample separation which computes the correct tip position from the recorded piezo height and the cantilever deflection. Other preprocessing steps correct for offsets or smoothen the data:

preprocessor key	description	details
compute_tip_position	Compute the tip-sample separation	*code reference*
correct_force_offset	Correct the force offset with an average baseline value	*code reference*
correct_split_approach_retract	Split the approach and retract curves (farthest point method)	*code reference*
correct_tip_offset	Correct the offset of the tip position	*code reference*
smooth_height	Smoothen height data	*code reference*

3.2 Models

Nanite comes with a predefined set of model functions, that are identified (in scripting as well as in the command line interface) via their model keys.

model key	description	details
hertz_cone	conical indenter (Hertz)	*code reference*
hertz_para	parabolic indenter (Hertz)	*code reference*
hertz_pyr3s	pyramidal indenter, three-sided (Hertz)	*code reference*
sneddon_spher	spherical indenter (Sneddon)	*code reference*
sneddon_spher_approx	spherical indenter (Sneddon, approximative)	*code reference*

These model functions can be used to fit experimental force-indentation data that have been preprocessed as described above.

3.3 Parameters

Besides the modeling parameters (e.g. Young's modulus or contact point), nanite allows to define an extensive set of fitting options, that are described in more detail in `nanite.fit.IndentationFitter`.

parameter	description
model_key	Key of the model function used
optimal_fit_edelta	Plateau search for Young's modulus
optimal_fit_num_samples	Number of points for plateau search
params_initial	Initial parameters
preprocessing	List of preprocessor keys
range_type	'absolute' for static range, 'relative cp' for dynamic range
range_x	Fitting range (min/max)
segment	Which segment to fit ('approach' or 'retract')
weight_cp	Suppression of residuals near contact point
x_axis	X-data used for fitting (defaults to 'top position')
y_axis	Y-data used for fitting (defaults to 'force')

3.4 Workflow

There are two ways to fit force-indentation curves with nanite: via the *command line interface (CLI)* or via Python scripting. The CLI does not require programming knowledge while Python-scripting allows fine-tuning and straightforward automation.

3.4.1 Command-line usage

First, setup up a fitting profile by running (e.g. in a command prompt on Windows).

```
nanite-setup-profile
```

This program will ask you to specify preprocessors, model parameters, and other fitting parameters. Simply enter the values via the keyboard and hit enter to let them be acknowledged. If you want to use the default values, simply hit enter without typing anything. A typical output will look like this:

```
Define preprocessing:
  1: compute_tip_position
  2: correct_force_offset
  3: correct_split_approach_retract
```

(continues on next page)

(continued from previous page)

```
  4: correct_tip_offset
  5: smooth_height
(currently '1,2,4'):

Select model number:
  1: hertz_cone
  2: hertz_para
  3: hertz_pyr3s
  4: sneddon_spher
  5: sneddon_spher_approx
(currently '5'):

Set fit parameters:
- initial value for E [Pa] (currently '3000.0'): 50
  vary E (currently 'True'):
- initial value for R [m] (currently '1e-5'): 18.64e-06
  vary R (currently 'False'):
- initial value for nu (currently '0.5'):
  vary nu (currently 'False'):
- initial value for contact_point [m] (currently '0.0'):
  vary contact_point (currently 'True'):
- initial value for baseline [N] (currently '0.0'):
  vary baseline (currently 'False'):

Select range type (absolute or relative):
(currently 'absolute'):

Select fitting interval:
left [µm] (currently '0.0'):
right [µm] (currently '0.0'):

Suppress residuals near contact point:
size [µm] (currently '0.5'): 2

Select training set:
training set (path or name) (currently 'zef18'):

Select rating regressor:
  1: AdaBoost
  2: Decision Tree
  3: Extra Trees
  4: Gradient Tree Boosting
  5: Random Forest
  6: SVR (RBF kernel)
  7: SVR (linear kernel)
(currently '3'):

Done. You may edit all parameters in '/home/user/.config/nanite/cli_profile.cfg'.
```

In this example, the only modifications of the default values are the initial value of the Young's modulus (50 Pa), the value for the tip radius (18.64 µm), and the suppression of residuals near the contact point with a ± 2 µm interval. When `nanite-setup-profile` is run again, it will use the values from the previous run as default values. The training set and rating regressor options are discussed in the *rating workflow*.

Finally, to perform the actual fitting, use the command-line script

```
nanite-fit data_path output_path
```

This command will recursively search the input folder `data_path` for data files, fit the data with the parameters in the profile, and write the statistics (*statistics.tsv*) and visualizations of the fits (multi-page TIFF file *plots.tif*, open with Fiji or the Windows Photo Viewer) to the directory `output_path`.

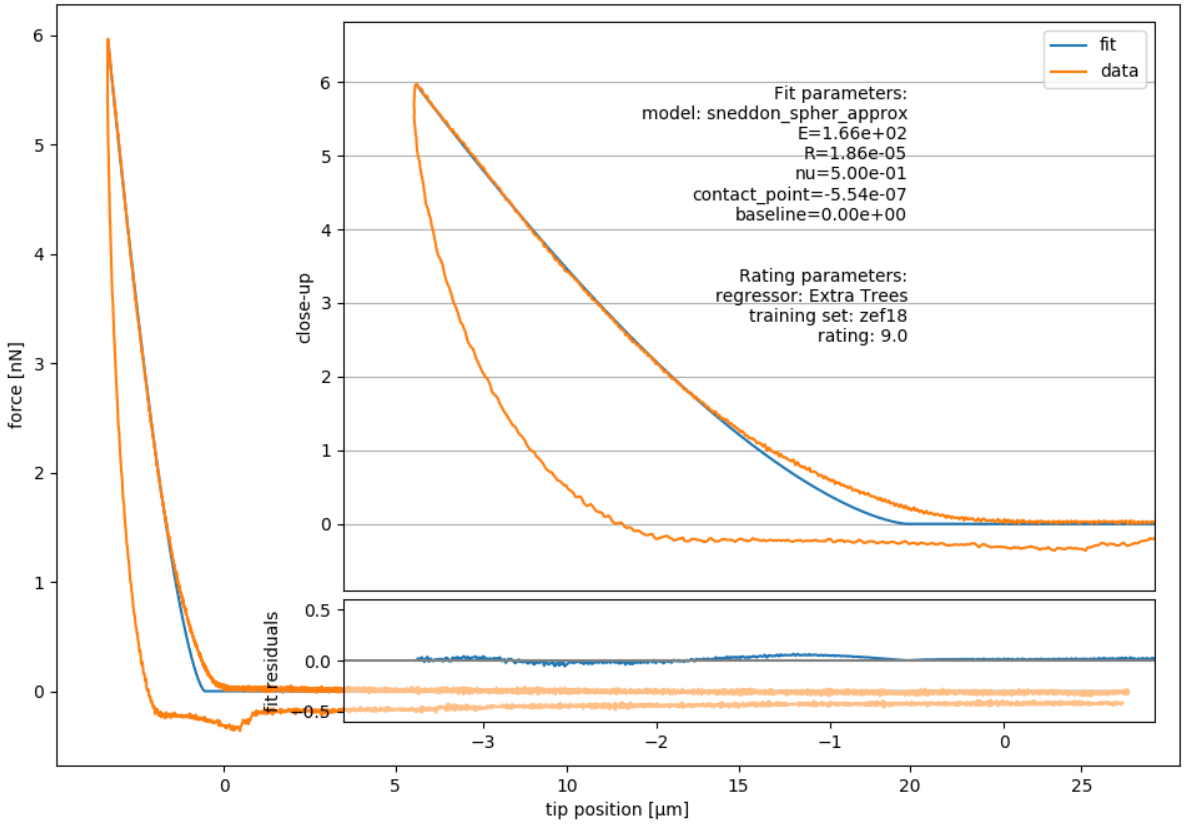


Fig. 3.1: Example image generated with `nanite-fit`. Note that the dataset is already rated with the default method "Extra Tree" and the default training set label "zef18". See *Rating workflow* for more information on rating.

3.4.2 Scripting usage

Using nanite in a Python script for data fitting is straight forward. First, load the data; `group` is an instance of *`nanite.IndentationGroup`*:

```
In [1]: import nanite

In [2]: group = nanite.load_group("data/force-save-example.jpk-force")
```

Second, obtain the first *`nanite.Indentation`* instance and apply the preprocessing:

```
In [3]: idnt = group[0]

In [4]: idnt.apply_preprocessing(["compute_tip_position",
   ...:                           "correct_force_offset",
```

(continues on next page)

(continued from previous page)

```
   ...:                                "correct_tip_offset"])
   ...:
```

Now, setup the model parameters:

```
In [5]: idnt.fit_properties["model_key"] = "sneddon_spher"

In [6]: params = idnt.get_initial_fit_parameters()

In [7]: params["E"].value = 50

In [8]: params["R"].value = 18.64e-06

In [9]: params.pretty_print()
Name               Value      Min      Max   Stderr     Vary     Expr
E                     50        0      inf     None     True     None
R              1.864e-05     -inf      inf     None    False     None
baseline               0     -inf      inf     None    False     None
contact_point          0     -inf      inf     None     True     None
nu                   0.5     -inf      inf     None    False     None
```

Finally, fit the model:

```
In [10]: idnt.fit_model(model_key="sneddon_spher", params_initial=params, weight_
 ↪cp=2e-6)

In [11]: idnt.fit_properties["params_fitted"].pretty_print()
Name               Value      Min      Max   Stderr     Vary     Expr
E                  165.8        0      inf   0.1802     True     None
R              1.864e-05     -inf      inf        0    False     None
baseline               0     -inf      inf        0    False     None
contact_point  -5.544e-07     -inf      inf 1.617e-09     True     None
nu                   0.5     -inf      inf        0    False     None
```

The fitting results are identical to those shown in figure 3.1 above.

Note that, amongst other things, preprocessing can also be specified directly in the *fit_model* function.

CHAPTER 4

Rating workflow

One of the main aims of nanite is to simplify data analysis by sorting out bad curves automatically based on a user defined rating scheme. Nanite allows to automate the rating process using machine learning, based on scikit-learn. In short, an estimator is trained with a sample dataset that was manually rated by a user. This estimator is then applied to new data and, in an optimal scenario, reproduces the rating scheme that the user intended when he rated the training dataset.

Nanite already comes with a default training set that is based on AFM data recorded for zebrafish spinal cord sections, called *zef18*. The original zef18 dataset is available on figshare *[MMG18]*. Download links:

- https://ndownloader.figshare.com/files/13481393

(SHA256: 63d89a8aa911a255fb4597b2c1801e30ea14810feef1bb42c11ef10f02a1d055)

With nanite, you can also create your own training set. The required steps to do so are described in the following.

4.1 Rating experimental data manually

In the rating step, experimental data are fitted and manually rated by the user. The raw data, the preprocessed data, the fit, all parameters, and the manual rating are then stored in an HDF5 (.h5) file.

First, set up a fitting profile using *nanite-setup-profile* if you have not already done so in the *fitting guide*. You can run the command `nanite-setup-profile` again to verify that all settings are correct.

To start manual rating, use the command *nanite-rate*. The first argument is a folder containing experimental force-indentation curves and the second argument is a path to a rating container (`nameXY.h5`). If the rating container already exists, new data will be appended (nothing is overridden).

```
nanite-rate path/to/data/directory path/to/nameXY.h5
```

This will open a graphical user interface that displays the preprocessed and fitted experimental data:

For the subsequent steps, it is irrelevant whether you create many small rating containers or one global rating container. Many small containers have the advantage that the effect of individual rating sessions could be analyzed separately, while a global rating container keeps all data in one place.

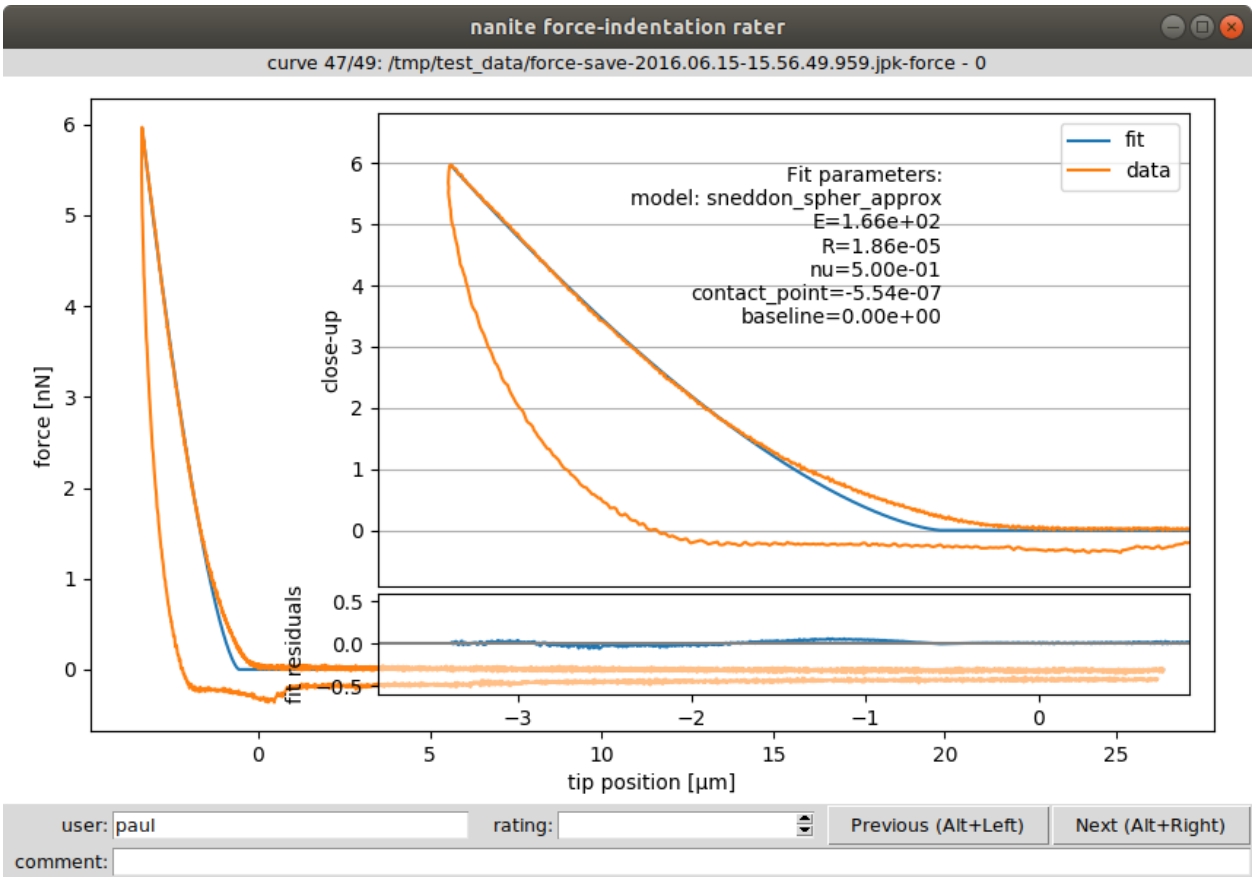


Fig. 4.1: Graphical user interface (GUI) for rating. The inset shows a close-up of the indentation part and the fitted parameters. The user name (defaults to login name) is used to assign a rating to a user (not mandatory). The rating (integer from 0/bad to 10/good or -1/invalid) and a comment can be defined for each curve. The shortcuts `ALT+Left` and `ALT+Right` can be used to navigate within the dataset while keeping the cursor focused in the *rating* field. While navigating, the data are stored in the rating container and the GUI can be closed without data loss.

4.2 Generating the training set

The training set consists only of the samples (features of each force-indentation curve) and the manual ratings. It is stored as a set of small text files on disk. As described earlier, nanite comes with the predefined *zef18* training set. In this step, a user-defined training set will be generated for use with nanite.

Use the command *nanite-generate-trainining-set* to convert the rating container(s) to a training set:

```
nanite-generate-trainining-set path/to/nameXY.h5 path/to/training_set/
```

This will create the folder `path/to/training_set/ts_nameXY` containing several text files, one for each feature and one for the manual rating.

4.3 Applying the training set

To apply the training set when rating curves with `nanite-fit`, you will have to update the profile using `nanite-setup-profile` again (see *fitting guide*). The relevant program output will look like this:

```
cmd>~: nanite-setup-profile

[...]

Select training set:
training set (path or name) (currently 'zef18'): path/to/training_set/ts_nameXY

Select rating regressor:
  1: AdaBoost
  2: Decision Tree
  3: Extra Trees
  4: Gradient Tree Boosting
  5: Random Forest
  6: SVR (RBF kernel)
  7: SVR (linear kernel)
(currently '3'):

Done. You may edit all parameters in '/home/user/.config/nanite/cli_profile.cfg'.
```

When running `nanite-fit data_path output_path` now, the new training set is used for rating. The new ratings are stored in `output_path/statistics.tsv` and can be used for further analysis, e.g. quality assessment or sorting.

If you would like to employ a user-defined training set in a Python script, you may do so by specifying the training set path as an argument to *nanite.Indentation.rate_quality*.

CHAPTER 5

Scripting examples

5.1 Approximating the Hertzian model with a spherical indenter

There is no closed form for the Hertzian model with a spherical indenter. The force F does not directly depend on the indentation depth δ, but has an indirect dependency via the radius of the circular contact area between indenter and sample a:

$$F = \frac{E}{1-\nu^2}\left(\frac{R^2+a^2}{2}\ln\left(\frac{R+a}{R-a}\right) - aR\right)$$
$$\delta = \frac{a}{2}\ln\left(\frac{R+a}{R-a}\right)$$

Here, E is the Young's modulus, R is the radius of the indenter, and ν is the Poisson's ratio of the probed material.

Because of this indirect dependency, fitting this model to experimental data can be time-consuming. Therefore, it is beneficial to approximate this model with a polynomial function around small values of δ/R using the Hertz model for a parabolic indenter as a starting point:

$$F = \frac{4}{3}\frac{E}{1-\nu^2}\sqrt{R}\delta^{3/2}\left(1 - \frac{1}{10}\frac{\delta}{R} - \frac{1}{840}\left(\frac{\delta}{R}\right)^2 + \frac{11}{15120}\left(\frac{\delta}{R}\right)^3 + \frac{1357}{6652800}\left(\frac{\delta}{R}\right)^4\right)$$

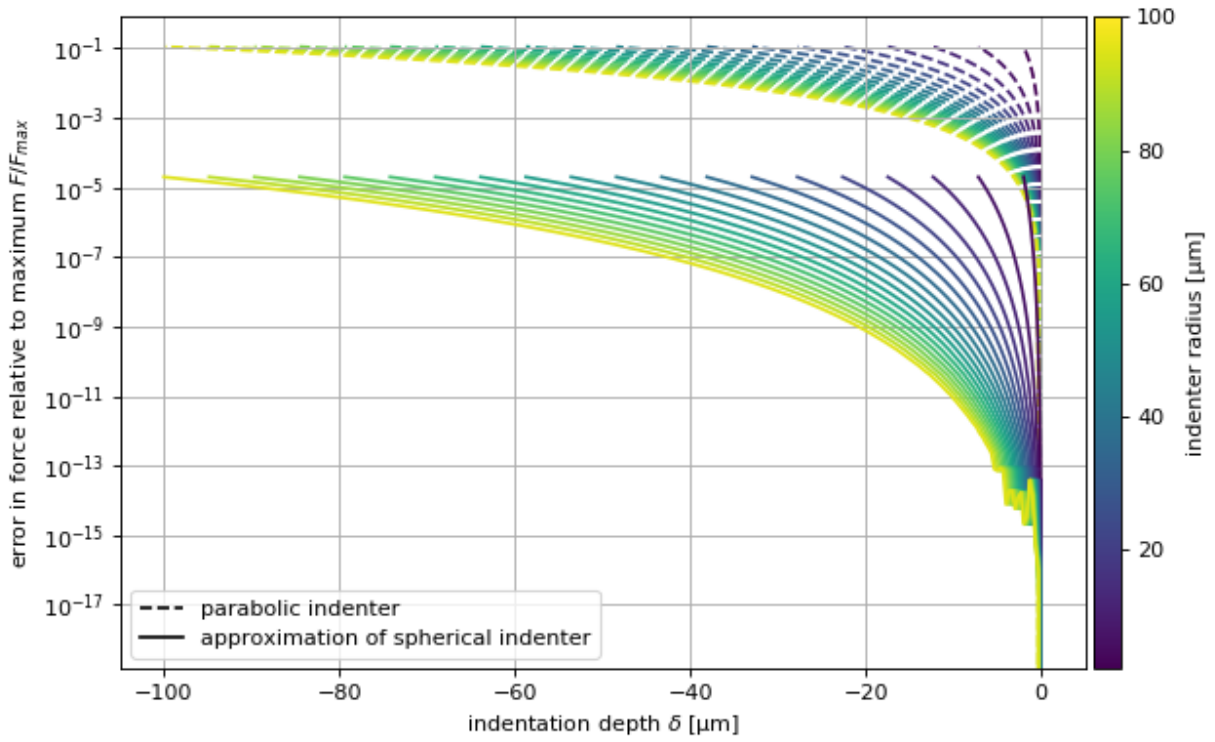
This example illustrates the error made with this approach. In nanite, the model for a spherical indenter has the identifier *"sneddon_spher"* and the approximate model has the identifier *"sneddon_spher_approx"*.

The plot shows the error for the parabolic indenter model *"hertz_para"* and for the approximation to the spherical indenter model. The maximum indentation depth is set to R. The error made by the approximation of the spherical indenter is more than four magnitudes lower than the maximum force during indentation.

`model_spherical_indenter.py`

```
import matplotlib.pylab as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib.lines import Line2D
import matplotlib as mpl
```

(continues on next page)



(continued from previous page)

```
import numpy as np

from nanite.model import models_available

# models
exact = models_available["sneddon_spher"]
approx = models_available["sneddon_spher_approx"]
para = models_available["hertz_para"]
# parameters
params = exact.get_parameter_defaults()
params["E"].value = 1000

# radii
radii = np.linspace(2e-6, 100e-6, 20)

# plot results
plt.figure(figsize=(8, 5))

# overview plot
ax = plt.subplot()
for ii, rad in enumerate(radii):
    params["R"].value = rad
    # indentation range
    x = np.linspace(0, -rad, 300)
    yex = exact.model(params, x)
    yap = approx.model(params, x)
    ypa = para.model(params, x)
    ax.plot(x*1e6, np.abs(yex - yap)/yex.max(),
```

(continues on next page)

(continued from previous page)

```
            color=mpl.cm.get_cmap("viridis")(ii/radii.size),
            zorder=2)
    ax.plot(x*1e6, np.abs(yex - ypa)/yex.max(), ls="--",
            color=mpl.cm.get_cmap("viridis")(ii/radii.size),
            zorder=1)

ax.set_xlabel("indentation depth $\delta$ [µm]")
ax.set_ylabel("error in force relative to maximum $F/F_{max}$")
ax.set_yscale("log")
ax.grid()

# legend
custom_lines = [Line2D([0], [0], color="k", ls="--"),
                Line2D([0], [0], color="k", ls="-"),
                ]
ax.legend(custom_lines, ['parabolic indenter',
                         'approximation of spherical indenter'])

divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="3%", pad=0.05)

norm = mpl.colors.Normalize(vmin=radii[0]*1e6, vmax=radii[-1]*1e6)
mpl.colorbar.ColorbarBase(ax=cax,
                          cmap="viridis",
                          norm=norm,
                          orientation='vertical',
                          label="indenter radius [µm]"
                          )

plt.tight_layout()
plt.savefig("model_spherical_indenter.jpg", dpi=120)
plt.show()
```

5.2 Fitting and rating

This example uses a force-indentation curve of a zebrafish spinal cord section to illustrate basic data fitting and rating with nanite. The dataset is part of a study on spinal cord stiffness in zebrafish [manuscript in preparation].

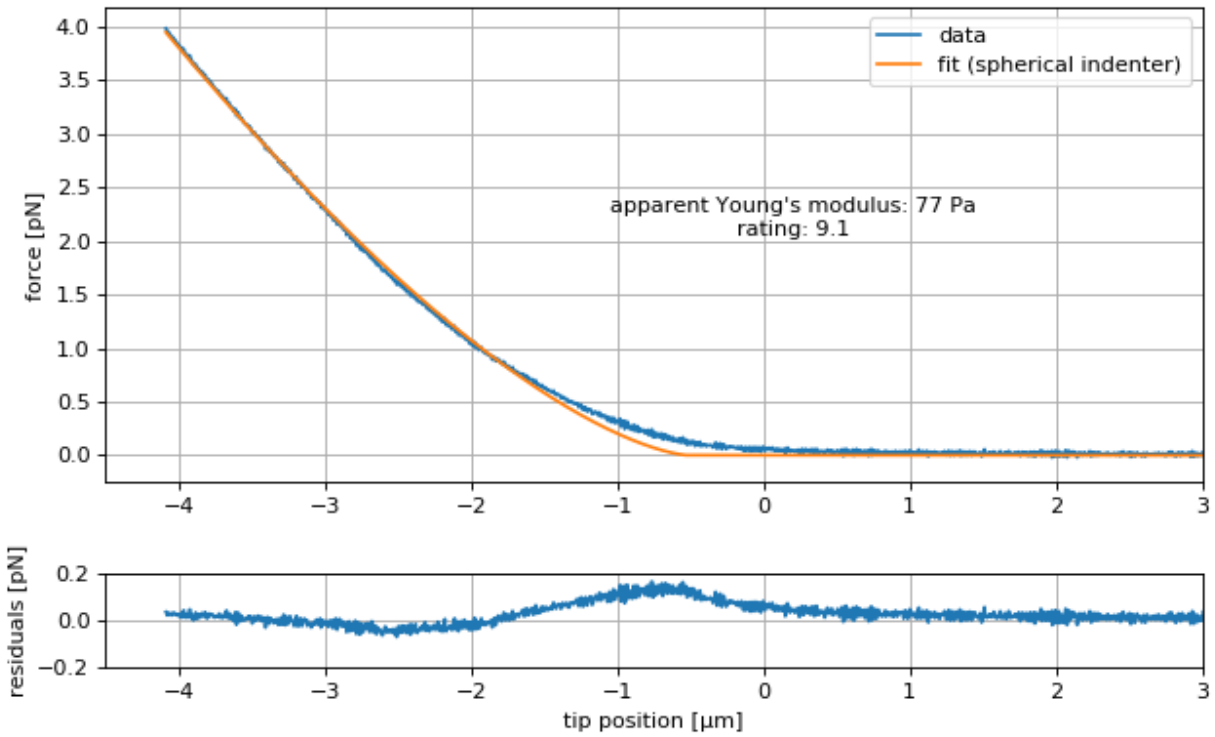
`fit_and_rate.py`

```
import matplotlib.gridspec as gridspec
import matplotlib.pylab as plt

import nanite

# load the data
group = nanite.load_group("data/zebrafish-head-section-gray-matter.jpk-force")
idnt = group[0]  # this is an instance of `nanite.Indentation`
# apply preprocessing
idnt.apply_preprocessing(["compute_tip_position",
                          "correct_force_offset",
                          "correct_tip_offset"])
# set the fit model ("sneddon_spher_approx" is faster than "sneddon_spher"
# and sufficiently accurate)
```

(continues on next page)



(continued from previous page)

```
idnt.fit_properties["model_key"] = "sneddon_spher_approx"
# get the initial fit parameters
params = idnt.get_initial_fit_parameters()
# set the correct indenter radius
params["R"].value = 18.64e-06
# perform the fit with the edited parameters
idnt.fit_model(params_initial=params)
# obtain the Young's modulus
emod = idnt.fit_properties["params_fitted"]["E"].value
# obtain a rating for the dataset
# (using default regressor and training set)
rate = idnt.rate_quality()

# overview plot
plt.figure(figsize=(8, 5))
gs = gridspec.GridSpec(2, 1, height_ratios=[5, 1])

ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1])

# only plot the approach part (`1` would be retract)
where_approach = idnt["segment"] == 0

# plot force-indentation data (nanite uses SI units)
ax1.plot(idnt["tip position"][where_approach] * 1e6,
         idnt["force"][where_approach] * 1e9,
         label="data")
ax1.plot(idnt["tip position"][where_approach] * 1e6,
```

(continues on next page)

(continued from previous page)

```
         idnt["fit"][where_approach] * 1e9,
         label="fit (spherical indenter)")
ax1.text(.2, 2.05,
         "apparent Young's modulus: {:.0f} Pa\n".format(emod)
         + "rating: {:.1f}".format(rate),
         ha="center")
ax1.legend()
# plot resiudals
ax2.plot(idnt["tip position"][where_approach] * 1e6,
         (idnt["force"] - idnt["fit"])[where_approach] * 1e9)

# update plot parameters
ax1.set_xlim(-4.5, 3)
ax1.set_ylabel("force [pN]")
ax1.grid()
ax2.set_xlim(-4.5, 3)
ax2.set_ylim(-.2, .2)
ax2.set_ylabel("residuals [pN]")
ax2.set_xlabel("tip position [µm]")
ax2.grid()

plt.tight_layout()
plt.show()
```

CHAPTER 6

Code reference

6.1 Module level aliases

For user convenience, the following objects are available at the module level.

class nanite.**Indentation**
alias of *nanite.indent.Indentation*

class nanite.**IndentationGroup**
alias of *nanite.group.IndentationGroup*

class nanite.**IndentationRater**
alias of nanite.rate.IndentationRater

class nanite.**QMap**
alias of *nanite.qmap.QMap*

nanite.**load_group**()
alias of *nanite.group.load_group*

6.2 Force-indentation data

class nanite.indent.**Indentation**(*idnt_data*)
Force-indentation

Parameters **idnt_data** (*nanite.read.IndentationData*) – Object holding the experimental data

apply_preprocessing(*preprocessing=None*)
Perform curve preprocessing steps

Parameters **preprocessing** (*list*) – A list of preprocessing method names that are stored in the *IndentationPreprocessor* class. If set to *None*, *self.preprocessing* will be used.

compute_emodulus_mindelta(*callback=None*)
Compute elastic modulus vs. mindelta

Parameters **callback** (*callable*) – A method that is called with the *emoduli* and *indentations* as the computation proceeds every five steps.

Returns **emoduli, indentations** – The fitted elastic moduli and at the corresponding maximal indentation depths.

Return type 1d ndarrays

Notes

The information about emodulus and mindelta is also stored in *self.fit_properties* with the keys "optimal_fit_E_array" and "optimal_fit_delta_array", if *self.fit_model* is called with the argument *search_optimal_fit* set to *True*.

estimate_contact_point_index()
Estimate the contact point

Contact point (CP) estimation is performed with two methods and that one which returns the smallest index is returned.

Method 1: baseline deviation

1. Obtain the baseline (initial 10% of the approach curve)
2. Compute average and maximum deviation of the baseline
3. The CP is the index of the approach curve where it exceeds twice of the maximum deviation

Method 2: sign of gradient

1. Perform a median filter on the approach curve
2. Compute the gradient
3. Cut off trailing 10 points from the gradient (noise)
4. The CP is the index of the gradient curve when the sign changes, measured from the point of maximal indentation.

If one of the methods fail, the index 0 is returned.

estimate_optimal_mindelta()
Estimate the optimal indentation depth

This is a convenience function that wraps around *compute_emodulus_mindelta* and *IndentationFitter.compute_opt_mindelta*.

export(*path*)
Saves the current data as tab separated values

fit_model(***kwargs*)
Fit the approach-retract data to a model function

Parameters

- **model_key** (*str*) – A key referring to a model in *nanite.model.models_available*
- **params_initial** (*instance of lmfit.Parameters or dict*) – Parameters for fitting. If not given, default parameters are used.
- **range_x** (*tuple of 2*) – The range for fitting, see *range_type* below.

- **range_type** (*str*) – One of:
  - **absolute:** Set the absolute fitting range in values given by the *x_axis*.
  - **relative cp:** In some cases it is desired to be able to fit a model only up until a certain indentation depth (tip position) measured from the contact point. Since the contact point is a fit parameter as well, this requires a two-pass fitting.
- **preprocessing** (*list of str*) – Preprocessing
- **segment** (*str*) – One of "approach" or "retract".
- **weight_cp** (*float*) – Weight the contact point region which shows artifacts that are difficult to model with e.g. Hertz.
- **optimal_fit_edelta** (*bool*) – Search for the optimal fit by varying the maximal indentation depth and determining a plateau in the resulting Young's modulus (fitting parameter "E").

get_initial_fit_parameters()

rate_quality(*regressor='Extra Trees'*, *training_set='zef18'*, *names=None*, *lda=None*)

Compute the quality of the obtained curve

Uses heuristic approaches to rate a curve.

Parameters

- **regressor** (*str*) – The regressor name used for rating.
- **training_set** (*str*) – A label for a training set shipped with nanite or a path to a training set.

Returns **rating** – A value between 0 and 10 where 0 is the lowest rating. If no fit has been performed, a rating of -1 is returned.

Return type float

Notes

The rating is cached based on the fitting hash (see *IndentationFitter._hash*).

reset()

Resets all data operations

data = None

All data in a Pandas DataFrame

fit_properties = None

Fitting results, see *Indentation.fit_model()*)

preprocessing = None

Default preprocessing steps steps, see *Indentation.apply_preprocessing()*.

6.3 Groups

class nanite.group.**IndentationGroup**(*path=None*, *callback=None*)

Group of Indentation

Parameters

- **path** (*str*) – The path to the data file. The data format is determined using the extension of the file and the data is loaded with the correct method.
- **callback** (*callable or None*) – A method that accepts a float between 0 and 1 to externally track the process of loading the data.

append(*item*)

index(*item*)

subgroup_with_path(*path*)
Return a subgroup with measurements matching *path*

nanite.group.**load_group**(*path*, *callback=None*)
Load indentation data from disk

Parameters

- **path** (*path-like*) – Path to experimental data
- **callback** (*callable or None*) – Callback function for tracking loading progress

Returns **group** – Indentation group with force-indentation data

Return type nanite.IndetationGroup

6.4 Loading data

nanite.read.**get_data_paths**(*path*)
Obtain a list of data files

Parameters **path** (*str or pathlib.Path*) – Path to a data file or a directory containing data files.

Returns **paths** – All supported data files found in *path*. If *path* is a file, *[pathlib.Path(path)]* is returned. If *path* has an unsupported extion, an empty list is returned.

Return type list of pathlib.Path

nanite.read.**get_data_paths_enum**(*path*, *skip_errors=False*)

nanite.read.**load_data**(*path*, *callback=None*)
Load data and return list of Indentation

nanite.read.**load_raw_data**(*path*, *callback=None*)
Load raw data

Parameters

- **path** (*str or pathlib.Path*) – Path to a data file or a directory containing data files. The data format is determined using the extension of the file.
- **callback** (*callable or None*) – A method that accepts a float between 0 and 1 to externally track the process of loading the data.
- **ret_indentation** (*bool*) – Return the indentation

Returns **data** – A measurements list that contains the data.

Return type list

nanite.read.**readers = [(<function load_jpk>, ['.jpk-force', '.jpk-force-map'])]**
All available readers and associated file extensions

nanite.read.**supported_extensions = ['.jpk-force', '.jpk-force-map']**
All supported file extensions

6.5 Preprocessing

exception nanite.preproc.**CannotSplitWarning**

class nanite.preproc.**IndentationPreprocessor**

static apply(*apret*, *preproc_names*)
Perform force-indentation preprocessing steps

Parameters

- **apret** (nanite.Indentation) – The afm data to preprocess
- **preproc_names** (*list*) – A list of names for static methods in *IndentationPreprocessor* that will be applied (in the order given).

Notes

This method is usually called from within the *Indentation* class instance. If you are using this class directly and apply it more than once, you might need to call *apret.reset()* before preprocessing a second time.

static available()
List available preprocessor names

static compute_tip_position(*apret*)
Compute the tip-sample separation

This computation correctly reproduces the column "Vertical Tip Position" as it is exported by the JPK analysis software with the checked option "Use Unsmoothed Height".

static correct_force_offset(*apret*)
Correct the force offset with an average baseline value

static correct_split_approach_retract(*apret*)
Split the approach and retract curves (farthest point method)

Approach and retract curves are defined by the microscope. When the direction of piezo movement is flipped, the force at the sample tip is still increasing. This can be either due to a time lag in the AFM system or due to a residual force acting on the sample due to the bent cantilever.

To repair this time lag, we append parts of the retract curve to the approach curve, such that the curves are split at the minimum height.

static correct_tip_offset(*apret*)
Correct the offset of the tip position

An estimate of the tip position is used to compute the contact point.

static smooth_height(*apret*)
Smoothen height data

For the columns "height (measured)" and "tip position", and for the approach and retract data separately, this method adds the columns "height (measured, smoothed)" and "tip position (smoothed)" to *self.data*.

nanite.preproc.**available_preprocessors = ['compute_tip_position', 'correct_force_offset',**
Available preprocessors

6.6 Modeling

6.6.1 Methods and constants

nanite.model.**get_init_parms**(*model_key*)
Get initial fit parameters for a model

nanite.model.**get_model_by_name**(*name*)
Convenience function to obtain a model by name instead of by key

nanite.model.**get_parm_name**(*model_key*, *parm_key*)
Get human readable parameter label

Parameters

- **model_key** (*str*) – The model key (e.g. "hertz_cone")
- **parm_key** (*str*) – The parameter key (e.g. "E")

Returns **parm_name** – The parameter name (e.g. "Young's Modulus")

Return type str

6.6.2 Models

Each model is implemented as a submodule in *nanite.model*. For instance *nanite.model.model_hertz_parabolic*. Each of these modules implements the following functions (which are not listed for each model in the subsections below):

nanite.model.model_submodule.**get_parameter_defaults**()
Return the default parameters of the model.

nanite.model.model_submodule.**model**()
Wrap the actual model for fitting.

nanite.model.model_submodule.**residual**()
Compute the residuals during fitting.

In addition, each submodule contains the following attributes:

nanite.model.model_submodule.**model_doc**
The doc-string of the model function.

nanite.model.model_submodule.**model_key**
The model key used in the command line interface and during scripting.

nanite.model.model_submodule.**model_name**
The name of the model.

nanite.model.model_submodule.**parameter_keys**
Parameter keywords of the model for higher-level applications.

nanite.model.model_submodule.**parameter_names**
Parameter names of the model for higher-level applications.

nanite.model.model_submodule.**parameter_units**
Parameter units for higher-level applications.

conical indenter (Hertz)

model key	hertz_cone
model name	conical indenter (Hertz)
model location	nanite.model.model_conical_indenter

nanite.model.model_conical_indenter.**hertz_conical**(*E*, *delta*, *alpha*, *nu*, *contact_point=0*, *baseline=0*)

Hertz model for a conical indenter

$$F = \frac{2\tan\alpha}{\pi}\frac{E}{1-\nu^2}\delta^2$$

Parameters

- **E** (*float*) – Young's modulus [N/m^2]
- **delta** (*1d ndarray*) – Indentation [m]
- **alpha** (*float*) – Half cone angle [degrees]
- **nu** (*float*) – Poisson's ratio
- **contact_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]
- **negindent** (*bool*) – If *True*, will assume that the indentation value(s) given by *delta* are negative and must be mutlitplied by -1.

Returns **F** – Force [N]

Return type float

Notes

These approximations are made by the Hertz model:

- The sample is isotropic.
- The sample is a linear elastic solid.
- The sample is extended infinitely in one half space.
- The indenter is not deformable.
- There are no additional interactions between sample and indenter.

Additional assumptions:

- infinitely sharp probe

References

Love (1939) *[LOV39]*

parabolic indenter (Hertz)

model key	hertz_para
model name	parabolic indenter (Hertz)
model location	nanite.model.model_hertz_parabolic

nanite.model.model_hertz_parabolic.**hertz_paraboloidal**(*E*, *delta*, *R*, *nu*, *contact_point=0*, *baseline=0*)

Hertz model for a paraboloidal indenter

$$F = \frac{4}{3}\frac{E}{1-\nu^2}\sqrt{R}\delta^{3/2}$$

Parameters

- **E** (*float*) – Young's modulus [N/m^2]
- **delta** (*1d ndarray*) – Indentation [m]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson's ratio
- **contact_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]
- **negindent** (*bool*) – If *True*, will assume that the indentation value(s) given by *delta* are negative and must be mutlitplied by -1.

Returns **F** – Force [N]

Return type float

Notes

The original model reads

$$F = \frac{4}{3}\frac{E}{1-\nu^2}\sqrt{2k}\delta^{3/2},$$

where k is defined by the paraboloid equation

$$\rho^2 = 4kz.$$

These approximations are made by the Hertz model:

- The sample is isotropic.
- The sample is a linear elastic solid.
- The sample is extended infinitely in one half space.
- The indenter is not deformable.
- There are no additional interactions between sample and indenter.

Additional assumptions:

- radius of spherical cell is larger than the indentation

References

Sneddon (1965) *[Sne65]*

pyramidal indenter, three-sided (Hertz)

model key	hertz_pyr3s
model name	pyramidal indenter, three-sided (Hertz)
model location	nanite.model.model_hertz_three_sided_pyramid

nanite.model.model_hertz_three_sided_pyramid.**hertz_three_sided_pyramid**(*E*, *delta*, *alpha*, *nu*, *contact_point=0*, *baseline=0*)

Hertz model for three sided pyramidal indenter

$$F = 0.887 \tan\alpha \cdot \frac{E}{1-\nu^2}\delta^2$$

Parameters

- **E** (*float*) – Young's modulus [N/m^2]
- **delta** (*1d ndarray*) – Indentation [m]
- **alpha** (*float*) – Face angle of the pyramid [degrees]
- **nu** (*float*) – Poisson's ratio
- **contact_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]
- **negindent** (*bool*) – If *True*, will assume that the indentation value(s) given by *delta* are negative and must be mutlitplied by -1.

Returns **F** – Force [N]

Return type float

Notes

These approximations are made by the Hertz model:

- The sample is isotropic.
- The sample is a linear elastic solid.
- The sample is extended infinitely in one half space.
- The indenter is not deformable.
- There are no additional interactions between sample and indenter.

References

Bilodeau et al. 1992 *[Bil92]*

spherical indenter (Sneddon)

model key	sneddon_spher
model name	spherical indenter (Sneddon)
model location	nanite.model.model_sneddon_spherical

nanite.model.model_sneddon_spherical.**delta_of_a**()
Compute indentation from contact area radius (wrapper)

nanite.model.model_sneddon_spherical.**get_a**()
Compute the contact area radius (wrapper)

nanite.model.model_sneddon_spherical.**hertz_spherical**()
Hertz model for Spherical indenter - modified by Sneddon

$$F = \frac{E}{1-\nu^2}\left(\frac{R^2+a^2}{2}\ln\left(\frac{R+a}{R-a}\right) - aR\right)$$
$$\delta = \frac{a}{2}\ln\left(\frac{R+a}{R-a}\right)$$

(a is the radius of the circular contact area between bead and sample.)

Parameters

- **E** (*float*) – Young's modulus [N/m^2]
- **delta** (*1d ndarray*) – Indentation [m]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson's ratio
- **contact_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]
- **negindent** (*bool*) – If *True*, will assume that the indentation value(s) given by *delta* are negative and must be multiplied by -1.

Returns **F** – Force [N]

Return type float

Notes

These approximations are made by the Hertz model:

- The sample is isotropic.
- The sample is a linear elastic solid.
- The sample is extended infinitely in one half space.
- The indenter is not deformable.
- There are no additional interactions between sample and indenter.

Additional assumptions:

- no surface forces

References

Sneddon (1965) *[Sne65]*

spherical indenter (Sneddon, approximative)

model key	sneddon_spher_approx
model name	spherical indenter (Sneddon, approximative)
model location	nanite.model.model_sneddon_spherical_approximation

nanite.model.model_sneddon_spherical_approximation.**hertz_sneddon_spherical_approx**(*E*, *delta*, *R*, *nu*, *contact_point=0*, *baseline=0*)

Hertz model for Spherical indenter - approximation

$$F = \frac{4}{3}\frac{E}{1-\nu^2}\sqrt{R}\delta^{3/2}\left(1 - \frac{1}{10}\frac{\delta}{R} - \frac{1}{840}\left(\frac{\delta}{R}\right)^2 + \frac{11}{15120}\left(\frac{\delta}{R}\right)^3 + \frac{1357}{6652800}\left(\frac{\delta}{R}\right)^4\right)$$

Parameters

- **E** (*float*) – Young's modulus [N/m^2]
- **delta** (*1d ndarray*) – Indentation [m]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson's ratio
- **contact_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]
- **negindent** (*bool*) – If *True*, will assume that the indentation value(s) given by *delta* are negative and must be mutlitplied by -1.

Returns **F** – Force [N]

Return type float

Notes

These approximations are made by the Hertz model:

- The sample is isotropic.
- The sample is a linear elastic solid.
- The sample is extended infinitely in one half space.

- The indenter is not deformable.
- There are no additional interactions between sample and indenter.

Additional assumptions:

- no surface forces

References

Sneddon (1965) *[Sne65]*, TODO

6.7 Fitting

exception nanite.fit.**FitDataError**

exception nanite.fit.**FitKeyError**

exception nanite.fit.**FitWarning**

class nanite.fit.**FitProperties**
Fit property manager class

Provide convenient access to fit properties as a dictionary and dynamically manage resets due to new initial parameters.

Dynamic properties include:

- set "params_initial" to *None* if the "model_key" changes
- remove all keys except those in *FP_DEFAULT* if a key that is in *FP_DEFAULT* changes (All other keys are considered to be obsolete fitting results).

Additional attributes:

- **"segment_bool": bool** *False* for "approach" and *True* for "retract"

reset()

restore(*props*)
update the dictionary without removing any keys

class nanite.fit.**IndentationFitter**(*data_set*, ***kwargs*)
Fit force-indentation curves

Parameters

- **model_key** (*str*) – A key referring to a model in *nanite.model.models_available*
- **params_initial** (*instance of lmfit.Parameters*) – Parameters for fitting. If not given, default parameters are used.
- **range_x** (*tuple of 2*) – The range for fitting, see *range_type* below.
- **range_type** (*str*) – One of:
  - **absolute:** Set the absolute fitting range in values given by the *x_axis*.
  - **relative cp:** In some cases it is desired to be able to fit a model only up until a certain indentation depth (tip position) measured from the contact point. Since the contact point is a fit parameter as well, this requires a two-pass fitting.
- **preprocessing** (*list of str*) – Preprocessing

- **segment** (*str*) – One of “approach” or “retract”.
- **weight_cp** (*float*) – Weight the contact point region which shows artifacts that are difficult to model with e.g. Hertz.
- **optimal_fit_edelta** (*bool*) – Search for the optimal fit by varying the maximal indentation depth and determining a plateau in the resulting Young’s modulus (fitting parameter “E”).
- **optimal_fit_num_samples** (*int*) – Number of samples to use for searching the optimal fit

compute_emodulus_vs_mindelta(*callback=None*)
Compute elastic modulus vs. minimal indentation curve

static compute_opt_mindelta(*emoduli*, *indentations*)
Determine the plateau of an emodulus-indentation curve

The following procedure is performed:

1. Smooth the emodulus data with a Butterworth filter
2. Label sequences that have similar values by binning into ten regions between the min and max.
3. Ignore sequences with emodulus that is smaller than the binning size.
4. Determine the longest sequence.

fit()
Fit the approach-retract data to a model function

get_initial_parameters(*data_set=None*, *model_key='hertz_para'*)
Get initial fit parameters for a specific model

nanite.fit.**obj2str**(*obj*)
String representation of an object for hashing

6.8 Rating

6.9 Quantitative maps

exception nanite.qmap.**DataMissingWarning**

class nanite.qmap.**QMap**(*path_or_dataset*, *callback=None*)
Quantitative force spectroscopy map handling

Parameters

- **path_or_dataset** (*str or nanite.IndentationGroup*) – The path to the data file. The data format is determined using the extension of the file and the data is loaded with the correct method.
- **callback** (*callable or None*) – A method that accepts a float between 0 and 1 to externally track the process of loading the data.

feat_data_min_height_measured_um(*idnt*)

feat_fit_youngs_modulus(*idnt*)

feat_meta_rating(*idnt*)

feat_meta_scan_order(*idnt*)

get_qmap (*feature*, *qmap_only=False*)
Return the quantitative map for a feature

Parameters

- **feature** (*str*) – Feature to compute map for (see *QMap.features*)
- **qmap_only** – Only return the quantitative map data, not the coordinates

Returns

- **x, y** (*1d ndarray*) – Only returned if *qmap_only* is False; Pixel grid coordinates along x and y
- **qmap** (*2d ndarray*) – Quantitative map

extent
extent (x1, x2, y1, y2) [µm]

features = None
Available features (see *nanite.qmap.available_features*)

get_coords
Get the qmap coordinates for each curve in *QMap.ds*

Parameters **which** (*str*) – "px" for pixels or "um" for microns.

group = None
Indentation data (instance of *nanite.IndentationGroup*)

shape
shape of the map [px]

nanite.qmap.**available_features = ['data min height', "fit young's modulus", 'meta rating',**
Available features for quantitative maps

CHAPTER 7

Changelog

List of changes in-between nanite releases.

7.1 version 0.9.1

- fix: *preprocessing* keyword not working in *Indentation.fit_model*
- docs: add another scripting example and minor improvements
- tests: increase coverage

7.2 version 0.9.0

- ref: remove legacy “discrete” feature type
- ref: renamed kwargs for *Indetation.rate_quality*
- ref: new method *nanite.load_group* for loading experimental data
- ref: new class read.data.IndentationData for managing data
- ref: replace dataset.IndentationDataSet with group.IndentationGroup to avoid ambiguities
- fix: add missing “zef18” training set
- fix: sample weight computation failed when a rating level was missing
- enh: add *nanite-generate-training-set* command line program
- tests: reduce warnings and increase coverage
- cleanup: old docs in nanite.rate.io
- docs: major update using helper extensions

7.3 version 0.8.0

- initial release

CHAPTER 8

Bilbliography

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[Bil92] G. G. Bilodeau. Regular pyramid punch problem. *Journal of Applied Mechanics*, 59(3):519, 1992. doi:10.1115/1.2893754.

[LOV39] A. E. H. LOVE. Boussinesq's problem for a rigid cone. *The Quarterly Journal of Mathematics*, os-10(1):161–175, 1939. doi:10.1093/qmath/os-10.1.161.

[MMG18] Paul Müller, Stephanie Möllmert, and Jochen Guck. Atomic force microscopy indentation data of zebrafish spinal cord sections. *Figshare*, 11 2018. doi:10.6084/m9.figshare.7297202.v1.

[Sne65] Ian N. Sneddon. The relation between load and penetration in the axisymmetric boussinesq problem for a punch of arbitrary profile. *International Journal of Engineering Science*, 3(1):47–57, may 1965. doi:10.1016/0020-7225(65)90019-4.

Python Module Index

n

Index