

---

# **nanite Documentation**

*Release 1.7.0*

**Paul Müller**

**Nov 20, 2020**



# CONTENTS

<b>1</b>	<b>Getting started</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	What is nanite? . . . . .	3
1.3	Supported file formats . . . . .	3
1.4	Use cases . . . . .	3
1.5	Basic usage . . . . .	4
1.5.1	How to cite . . . . .	4
<b>2</b>	<b>Command-line interface</b>	<b>5</b>
2.1	nanite-setup-profile . . . . .	5
2.2	nanite-fit . . . . .	5
2.3	nanite-rate . . . . .	5
2.4	nanite-generate-training-set . . . . .	6
<b>3</b>	<b>Fitting guide</b>	<b>7</b>
3.1	Preprocessors . . . . .	7
3.2	Models . . . . .	7
3.3	Parameters . . . . .	8
3.4	Workflow . . . . .	8
3.4.1	Command-line usage . . . . .	8
3.4.2	Scripting usage . . . . .	11
<b>4</b>	<b>Rating workflow</b>	<b>13</b>
4.1	Rating experimental data manually . . . . .	13
4.2	Generating the training set . . . . .	15
4.3	Applying the training set . . . . .	15
<b>5</b>	<b>Scripting examples</b>	<b>17</b>
5.1	Approximating the Hertzian model with a spherical indenter . . . . .	17
5.2	Fitting and rating . . . . .	19
<b>6</b>	<b>Developer guide</b>	<b>23</b>
6.1	How to contribute . . . . .	23
6.2	Updating the documentation . . . . .	23
6.3	Writing model functions . . . . .	23
6.3.1	Getting started . . . . .	24
6.3.2	Ancillary parameters . . . . .	25
<b>7</b>	<b>Code reference</b>	<b>27</b>
7.1	Module level aliases . . . . .	27
7.2	Force-indentation data . . . . .	27

7.3	Groups . . . . .	30
7.4	Loading data . . . . .	31
7.5	Preprocessing . . . . .	31
7.6	Modeling . . . . .	32
	7.6.1 Methods and constants . . . . .	32
	7.6.2 Models . . . . .	34
7.7	Fitting . . . . .	43
7.8	Rating . . . . .	45
	7.8.1 Features . . . . .	45
	7.8.2 Rater . . . . .	47
	7.8.3 Regressors . . . . .	49
	7.8.4 Manager . . . . .	49
7.9	Quantitative maps . . . . .	50
<b>8</b>	<b>Changelog</b>	<b>53</b>
8.1	version 1.7.0 . . . . .	53
8.2	version 1.6.3 . . . . .	53
8.3	version 1.6.2 . . . . .	53
8.4	version 1.6.1 . . . . .	53
8.5	version 1.6.0 . . . . .	53
8.6	version 1.5.5 . . . . .	54
8.7	version 1.5.4 . . . . .	54
8.8	version 1.5.3 . . . . .	54
8.9	version 1.5.2 . . . . .	54
8.10	version 1.5.1 . . . . .	54
8.11	version 1.5.0 . . . . .	54
8.12	version 1.4.1 . . . . .	54
8.13	version 1.4.0 . . . . .	55
8.14	version 1.3.0 . . . . .	55
8.15	version 1.2.4 . . . . .	55
8.16	version 1.2.3 . . . . .	55
8.17	version 1.2.2 . . . . .	55
8.18	version 1.2.1 . . . . .	55
8.19	version 1.2.0 . . . . .	56
8.20	version 1.1.2 . . . . .	56
8.21	version 1.1.1 . . . . .	56
8.22	version 1.1.0 . . . . .	56
8.23	version 1.0.1 . . . . .	56
8.24	version 1.0.0 . . . . .	56
8.25	version 0.9.3 . . . . .	56
8.26	version 0.9.2 . . . . .	57
8.27	version 0.9.1 . . . . .	57
8.28	version 0.9.0 . . . . .	57
8.29	version 0.8.0 . . . . .	57
<b>9</b>	<b>Bibliography</b>	<b>59</b>
<b>10</b>	<b>Indices and tables</b>	<b>61</b>
	<b>Bibliography</b>	<b>63</b>
	<b>Python Module Index</b>	<b>65</b>
	<b>Index</b>	<b>67</b>

Nanite is a Python library for loading, fitting, and rating AFM force-distance data of cells and tissues. This is the documentation of nanite version 1.7.0.



## 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-distance 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 (à la *scikit-learn*), which allows to automatically determine the quality of a force-distance 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-distance data analysis.

### 1.3 Supported file formats

Nanite relies on the *afmformats* package. A list of supported file formats can be found [here](#).

### 1.4 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-distance maps. Here are a few usage examples of nanite:

- You would like to automate your data analysis pipeline from loading force-distance 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-distance data. This is possible with the `nanite.QMap` class.
- You would like to sort force-distance 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-distance maps. For an overview, see [Rating workflow](#).

## 1.5 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.jpik-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]: 9.060746150910978
```

You can find more examples in the [examples](#) section.

### 1.5.1 How to cite

If you use nanite in a scientific publication, please cite Müller et al., *BMC Bioinformatics* (2019) [MAM+19].



## 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-distance 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-distance data
out_dir	results directory

### 2.3 nanite-rate

Manually rate (the fit to) AFM force-distance data. A graphical user interface allows to rate and comment on each force-distance curve. The fits and the raw 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-distance 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

## FITTING GUIDE

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

### 3.1 Preprocessors

Prior to data analysis, a force-distance curve 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	<i>code reference</i>
correct_force_offset	Correct the force offset with an average baseline value	<i>code reference</i>
correct_split_approach_retract	Split the approach and retract curves (farthest point method)	<i>code reference</i>
correct_tip_offset	Correct the offset of the tip position	<i>code reference</i>
smooth_height	Smoothen height data	<i>code reference</i>

### 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)	<i>code reference</i>
hertz_para	parabolic indenter (Hertz)	<i>code reference</i>
hertz_pyr3s	pyramidal indenter, three-sided (Hertz)	<i>code reference</i>
sneddon_spher	spherical indenter (Sneddon)	<i>code reference</i>
sneddon_spher_approx	spherical indenter (Sneddon, approximative)	<i>code reference</i>

These model functions can be used to fit experimental force-distance 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-distance 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 straight-forward 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
 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:
```

(continues on next page)

(continued from previous page)

```

- 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  $\pm 2$  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`.

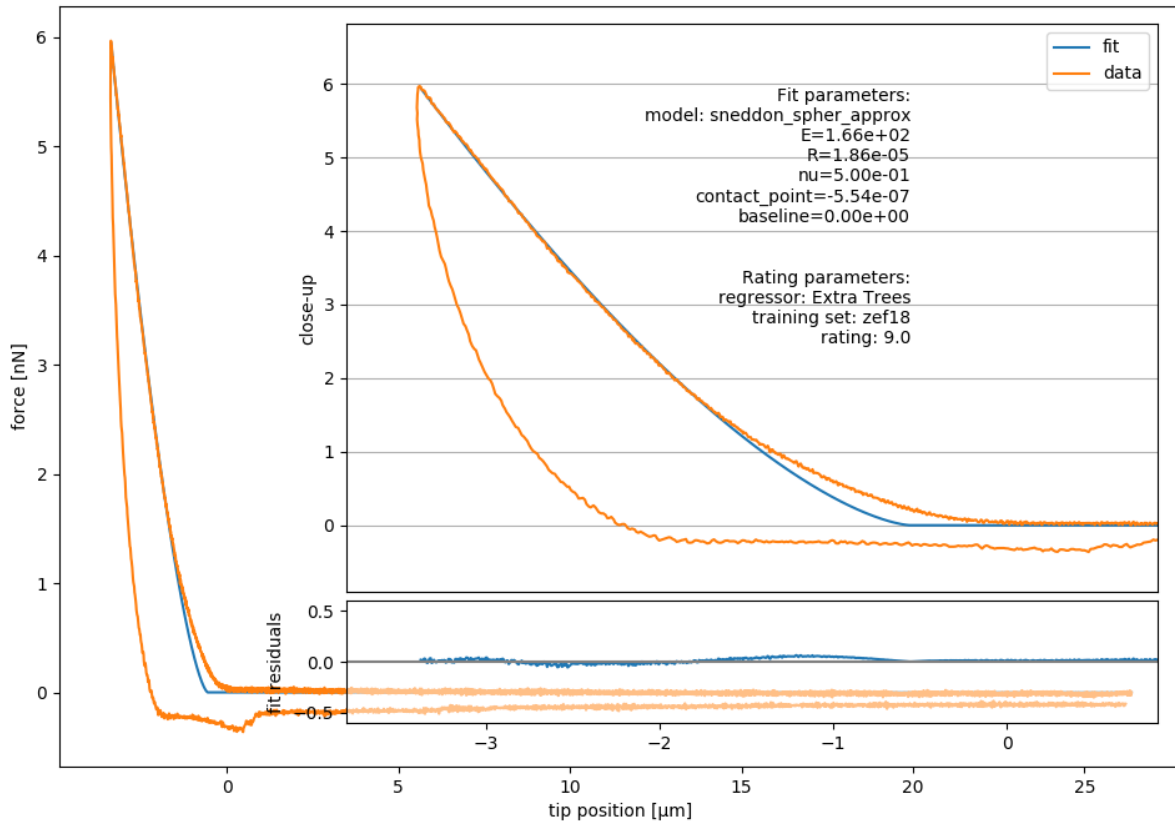


Fig. 3.1: Example image generated with `nanite-fit`. Note that the dataset is already rated with the default method “Extra Trees” and the default training set label “zef18”. See [Rating workflow](#) for more information on rating.

### 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.jpj-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",
...:                             "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 Brute_Step
E              50         0      inf     None   True   None   None
R          1.864e-05      0      inf     None  False   None   None
baseline         0      -inf     inf     None   True   None   None
contact_point    0      -inf     inf     None   True   None   None
nu              0.5         0      0.5     None  False   None   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 Brute_Step
E            165.8         0      inf   0.1804   True   None   None
R          1.864e-05      0      inf      0  False   None   None
baseline    -1.678e-13     -inf     inf  2.318e-13   True   None   None
contact_point -9.419e-07     -inf     inf  1.623e-09   True   None   None
nu              0.5         0      0.5      0  False   None   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.





## 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. For a more detailed analysis, please refer to [MAM+19].

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 online [MMG18]. Download links:<sup>1</sup>

- <https://ndownloader.figshare.com/files/13481393>
- <https://zenodo.org/record/1551200/files/zef18.h5>
- <https://b2share.eudat.eu/api/files/bf481c9b-14ff-47b1-baf5-e569d0199be6/zef18.h5>

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 a rating container (an HDF5 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-distance 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.

---

<sup>1</sup> The SHA256 checksum of *zef18.h5* is 63d89a8aa911a255fb4597b2c1801e30ea14810feef1bb42c11ef10f02a1d055.

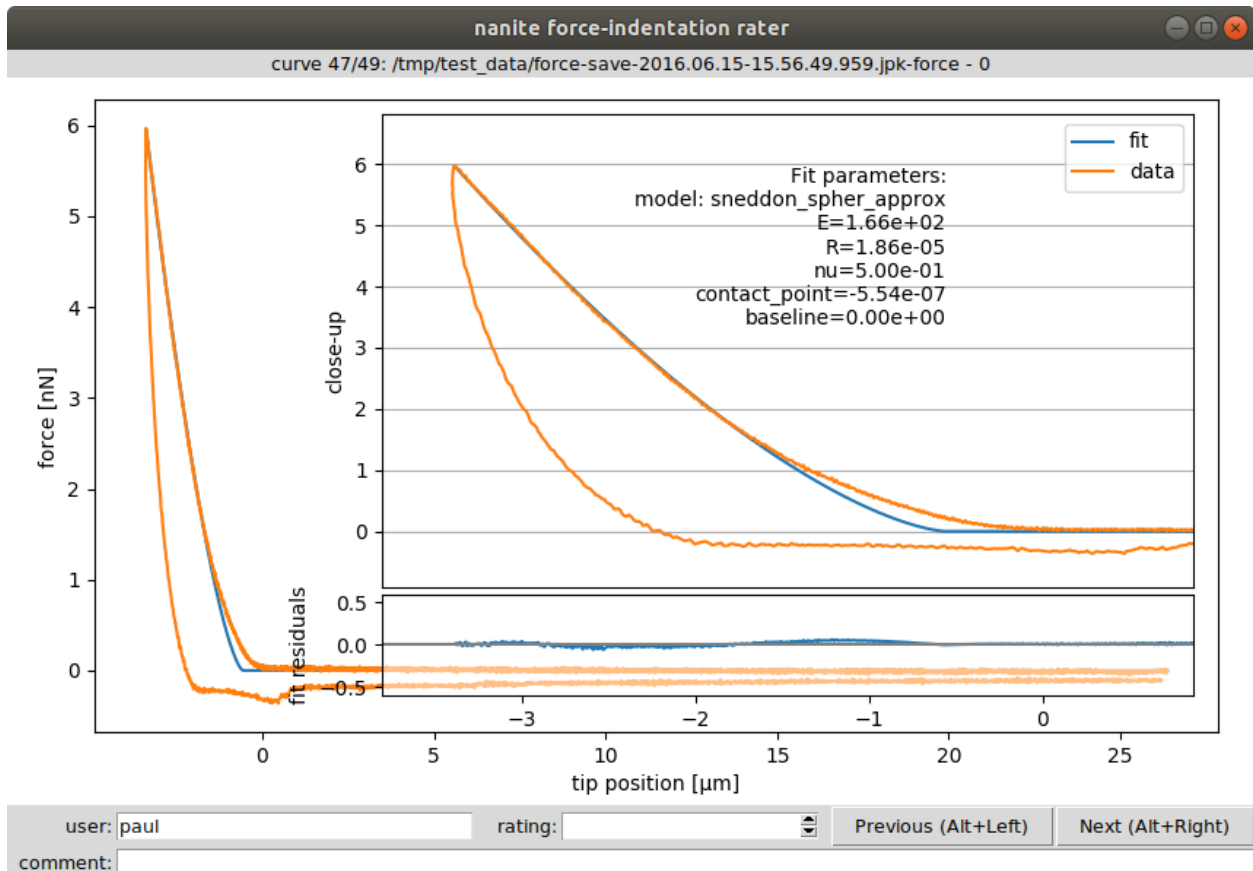


Fig. 4.1: Graphical user interface (GUI) for rating. The inset shows a close-up of the indentation part and the fitted parameters. The user name (defaults to login name) is used to assign a rating to a user (not mandatory). The rating (integer from 0/bad to 10/good or -1/invalid) and a comment can be 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-distance 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-training-set` to convert the rating container(s) to a training set:

```
nanite-generate-training-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:

```
[...]

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`.



## 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  $\delta$ , but has an indirect dependency via the radius of the circular contact area between indenter and sample  $a$  [Sne65]:

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

$$\delta = \frac{a}{2} \ln \left( \frac{R + a}{R - a} \right)$$

Here,  $E$  is the Young's modulus,  $R$  is the radius of the indenter, and  $\nu$  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  $\delta/R$  using the Hertz model for a parabolic indenter as a starting point [Dob18]:

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

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.

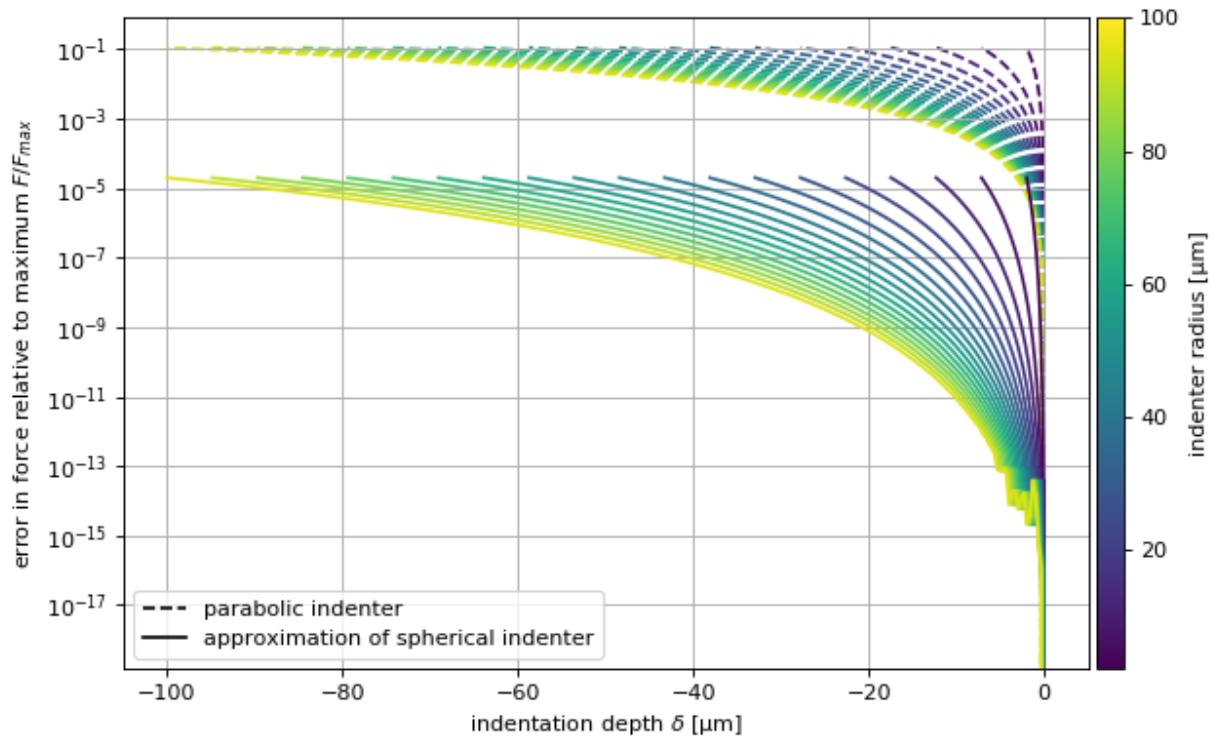
model\_spherical\_indenter.py

```

1 import matplotlib.pyplot as plt
2 from mpl_toolkits.axes_grid1 import make_axes_locatable
3 from matplotlib.lines import Line2D
4 import matplotlib as mpl
5 import numpy as np
6
7 from nanite.model import models_available
8
9 # models
10 exact = models_available["sneddon_spher"]
11 approx = models_available["sneddon_spher_approx"]
12 para = models_available["hertz_para"]
13 # parameters

```

(continues on next page)



(continued from previous page)

```

14 params = exact.get_parameter_defaults()
15 params["E"].value = 1000
16
17 # radii
18 radii = np.linspace(2e-6, 100e-6, 20)
19
20 # plot results
21 plt.figure(figsize=(8, 5))
22
23 # overview plot
24 ax = plt.subplot()
25 for ii, rad in enumerate(radii):
26     params["R"].value = rad
27     # indentation range
28     x = np.linspace(0, -rad, 300)
29     yex = exact.model(params, x)
30     yap = approx.model(params, x)
31     ypa = para.model(params, x)
32     ax.plot(x*1e6, np.abs(yex - yap)/yex.max(),
33            color=mpl.cm.get_cmap("viridis")(ii/radii.size),
34            zorder=2)
35     ax.plot(x*1e6, np.abs(yex - ypa)/yex.max(), ls="--",
36            color=mpl.cm.get_cmap("viridis")(ii/radii.size),
37            zorder=1)
38
39 ax.set_xlabel(r"indentation depth  $\delta$  [ $\mu\text{m}$ ]")
40 ax.set_ylabel("error in force relative to maximum  $F/F_{\text{max}}$ ")
41 ax.set_yscale("log")

```

(continues on next page)

(continued from previous page)

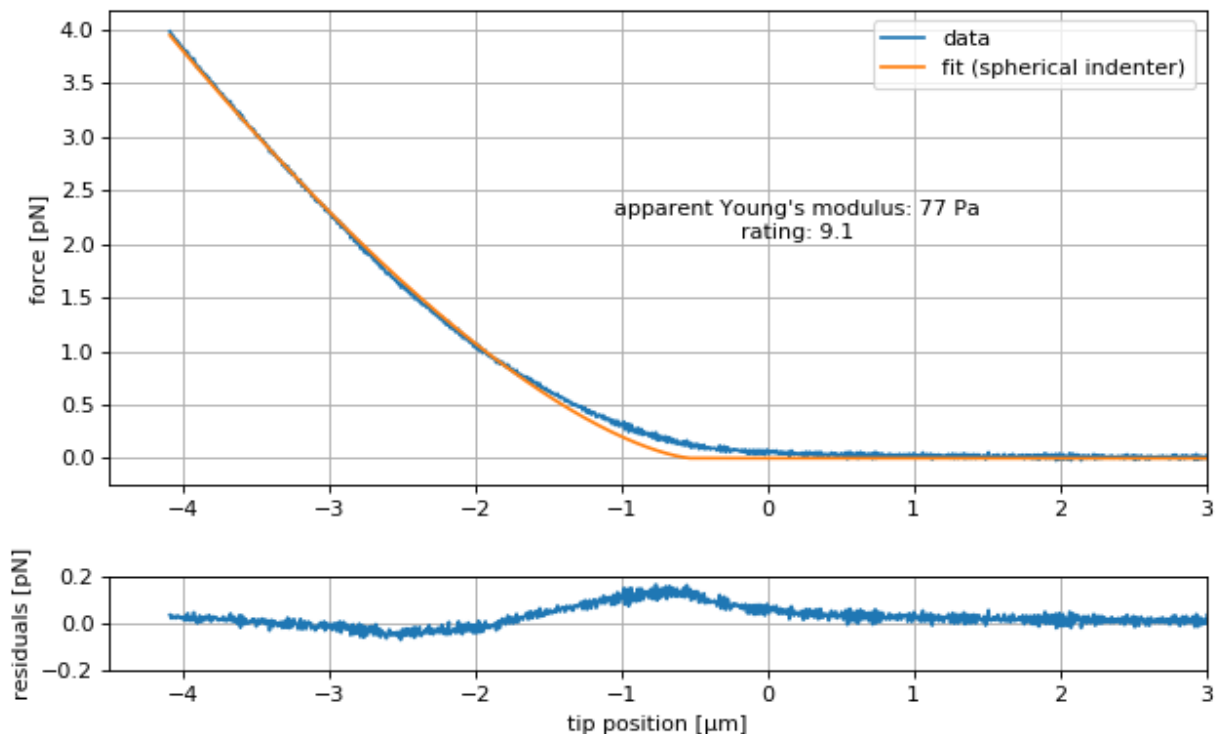
```

42 ax.grid()
43
44 # legend
45 custom_lines = [Line2D([0], [0], color="k", ls="--"),
46                  Line2D([0], [0], color="k", ls="--"),
47                  ]
48 ax.legend(custom_lines, ['parabolic indenter',
49                          'approximation of spherical indenter'])
50
51 divider = make_axes_locatable(ax)
52 cax = divider.append_axes("right", size="3%", pad=0.05)
53
54 norm = mpl.colors.Normalize(vmin=radii[0]*1e6, vmax=radii[-1]*1e6)
55 mpl.colorbar.ColorbarBase(ax=cax,
56                           cmap=mpl.cm.viridis,
57                           norm=norm,
58                           orientation='vertical',
59                           label="indenter radius [μm]"
60                           )
61
62 plt.tight_layout()
63 plt.show()

```

## 5.2 Fitting and rating

This example uses a force-distance 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 [MKH+19].



fit\_and\_rate.py

```

1 import matplotlib.gridspec as gridspec
2 import matplotlib.pyplot as plt
3
4 import nanite
5
6 # load the data
7 group = nanite.load_group("data/zebrafish-head-section-gray-matter.jpik-force")
8 idnt = group[0] # this is an instance of `nanite.Indentation`
9 # apply preprocessing
10 idnt.apply_preprocessing(["compute_tip_position",
11                          "correct_force_offset",
12                          "correct_tip_offset"])
13 # set the fit model ("sneddon_spher_approx" is faster than "sneddon_spher"
14 # and sufficiently accurate)
15 idnt.fit_properties["model_key"] = "sneddon_spher_approx"
16 # get the initial fit parameters
17 params = idnt.get_initial_fit_parameters()
18 # set the correct indenter radius
19 params["R"].value = 18.64e-06
20 # perform the fit with the edited parameters
21 idnt.fit_model(params_initial=params)
22 # obtain the Young's modulus
23 emod = idnt.fit_properties["params_fitted"]["E"].value
24 # obtain a rating for the dataset
25 # (using default regressor and training set)
26 rate = idnt.rate_quality()
27
28 # overview plot
29 plt.figure(figsize=(8, 5))
30 gs = gridspec.GridSpec(2, 1, height_ratios=[5, 1])
31
32 ax1 = plt.subplot(gs[0])
33 ax2 = plt.subplot(gs[1])
34
35 # only plot the approach part (`1` would be retract)
36 where_approach = idnt["segment"] == 0
37
38 # plot force-distance data (nanite uses SI units)
39 ax1.plot(idnt["tip position"][where_approach] * 1e6,
40         idnt["force"][where_approach] * 1e9,
41         label="data")
42 ax1.plot(idnt["tip position"][where_approach] * 1e6,
43         idnt["fit"][where_approach] * 1e9,
44         label="fit (spherical indenter)")
45 ax1.text(.2, 2.05,
46         "apparent Young's modulus: {:.0f} Pa\n".format(emod)
47         + "rating: {:.1f}".format(rate),
48         ha="center")
49 ax1.legend()
50 # plot residuals
51 ax2.plot(idnt["tip position"][where_approach] * 1e6,
52         (idnt["force"] - idnt["fit"])[where_approach] * 1e9)
53
54 # update plot parameters
55 ax1.set_xlim(-4.5, 3)
56 ax1.set_ylabel("force [pN]")

```

(continues on next page)



(continued from previous page)

```
57 ax1.grid()
58 ax2.set_xlim(-4.5, 3)
59 ax2.set_ylim(-.2, .2)
60 ax2.set_ylabel("residuals [pN]")
61 ax2.set_xlabel("tip position [ $\mu\text{m}$ ]")
62 ax2.grid()
63
64 plt.tight_layout()
65 plt.show()
```



## DEVELOPER GUIDE

### 6.1 How to contribute

Contributions via pull requests are very welcome. Just fork the “master” branch, make your changes, and create a pull request back to “master” with a descriptive title and an explanation of what you have done. If you decide to contribute code, please

1. properly document your code (in-line comments as well as doc strings),
2. ensure code quality with `flake8` and `autopep8`,
3. write test functions for `pytest` (aim for 100% code coverage),
4. update the changelog (for new features, increment to the next minor release; for small changes or bug fixes, increment the patch number)

### 6.2 Updating the documentation

The documentation is stored in the `docs` directory of the repository and is built using `sphinx`.

To build the documentation, first install the build requirements by running this in the `docs` directory:

```
pip install -r requirements.txt`
```

You can now build the documentation with

```
sphinx-build . _build
```

Open the file `_build/index.html` in your web browser to view the result.

### 6.3 Writing model functions

You are here because you would like to write a new model function for `nanite`. Note that all model functions implemented in `nanite` are consequently available in `PyJibe` as well.

### 6.3.1 Getting started

First, create a Python file `model_unique_name.py` which will be the home of your new model (make sure the name starts with `model_`). Place the file in the following location: `nanite/model/model_unique_name.py`. Your file should at least contain the following:

```
import lmfit
import numpy as np

def get_parameter_defaults():
    """Return the default model parameters"""
    # The order of the parameters must match the order
    # of 'parameter_names' and 'parameter_keys'.
    params = lmfit.Parameters()
    params.add("E", value=3e3, min=0)
    params.add("contact_point", value=0)
    params.add("baseline", value=0)
    return params

def your_model_name(delta, E, contact_point=0, baseline=0):
    r"""A brief model description

    A more elaborate model description with a formula.

    .. math::

        F = \frac{4}{3}
            \frac{E}{\delta^{3/2}}

    Parameters
    -----
    delta: 1d ndarray
        Indentation [m]
    E: float
        Young's modulus [N/m2]
    contact_point: float
        Indentation offset [m]
    baseline: float
        Force offset [N]

    Returns
    -----
    F: float
        Force [N]

    Notes
    -----
    Here you can add more information about the model.

    References
    -----
    Please give proper references for your model (e.g. publications or
    arXiv manuscripts. You can do so by editing the "docs/nanite.bib"
    file and cite it like so:
    Sneddon (1965) :cite:`Sneddon1965`
```

(continues on next page)

(continued from previous page)

```

"""
# this is a convention to avoid computing the root of negative values
root = contact_point - delta
pos = root > 0
# this is the model output
out = np.zeros_like(delta)
out[pos] = 4/3 * E * root[pos]**(3/2)
# add the baseline
return out + baseline

model_doc = your_model_name.__doc__
model_func = your_model_name
model_key = "unique_model_key"
model_name = "short model name"
parameter_keys = ["E", "contact_point", "baseline"]
parameter_names = ["Young's Modulus", "Contact Point", "Force Baseline"]
parameter_units = ["Pa", "m", "N"]
valid_axes_x = ["tip position"]
valid_axes_y = ["force"]

```

Once you have created this file, you have to register it in nanite by adding the line

```
from . import model_unique_name # noqa: F401
```

at the top in the file `nanite/model/__init__.py`. That's it!

A few things should be noted:

- When designing your model parameters, always use SI units.
- Always include a model formula. You can test whether it renders correctly by building the documentation (see above) and checking whether your model shows up properly in the code reference.
- Fitting parameters should not contain spaces. Only use characters that are allowed in Python variable names.
- Since fitting is based on `lmfit`, you may define [mathematical constraints](#) in `get_parameter_defaults`. However, if possible, try to solve your particular problem with ancillaries (see below), a concept that is easier to understand.

Now it is time for a quick sanity check:

```
from nanite import model
assert "unique_model_key" in model.models_available
```

### 6.3.2 Ancillary parameters

For more elaborate models, you might need additional parameters from the `nanite.indent.Indentation` instance. This is where ancillary parameters come into play.

You can define an arbitrary number of ancillary parameters in your `model_unique_name.py` file:

```
def compute_ancillaries(idnt):
    """Compute ancillaries for my model

    Parameters
    -----

```

(continues on next page)

```

idnt: nanite.indent.Indentation
    Indentation dataset from which to extract the ancillary
    parameters.

Returns
-----
example: dict
    Dictionary with ancillary parameters. In this example:

    - "force_range": total force range covered by approach and retract
    """
# You have access to the initial fit parameters (including a
# good contact point estimate) with this line:
parms = idnt.get_initial_fit_parameters(model_key=model_key,
                                       model_ancillaries=False)

# You can access individual columns...
force = idnt.data["force"]
segment = idnt.data["segment"] # `False` for approach; `True` for retract
tip_position = idnt.data["tip position"]

# ...and segments
force_approach = force[~segment] # equivalent to force[segment == False]
force_retract = force[segment]

# Initialize ancillary dictionary.
anc_dict = dict()

# This is the exemplary force parameter
anc_dict["force_range"] = np.ptp(force)

return anc_dict

# And below the other `parameter_keys` etc.:
parameter_anc_keys = ["force_range"]
parameter_anc_names = ["Overall peak-to-peak force"]
parameter_anc_units = ["N"]

```

#### You should know:

- If an ancillary parameter key matches that of a fitting parameter (defined in `get_parameter_defaults` above), then the ancillary parameter can be used as an initial value for fitting (see `nanite.fit.guess_initial_parameters()`).
- If `compute_ancillaries` does not know how to compute a certain parameter, it should set it to `np.nan` instead of `None` (compatibility with PyJibe).
- If you would like to define an ancillary parameter that depends on a successful fit, you could first check against `idnt.fit_properties["success"]` and then compute your parameter (else set it to `np.nan`).

## CODE REFERENCE

### 7.1 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
```

### 7.2 Force-indentation data

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

    Parameters idnt_data (nanite.read.IndentationData) – Object holding the experi-
        mental 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)
        Elastic modulus in dependency of maximum indentation

        The fitting interval is varied such that the maximum indentation depth ranges from the lowest tip position
        to the estimated contact point. For each interval, the current model is fitted and the elastic modulus is
        extracted.

        Parameters callback (callable) – A method that is called with the emoduli and indenta-
            tions as the computation proceeds every five steps.
```

**Returns** `emoduli, indentations` – The fitted elastic moduli 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 involves a preprocessing step where the force data are transformed into gradient space (to account for a slope in the approach curve) and a subsequent analysis with two different methods to determine when the gradient changes significantly enough to qualify for a CP. Of those two methods, the one which yields the smallest index (measured from the beginning of the approach curve) is returned. If one of the methods fail, then a fit function with a constant and linear part is used to determine the CP.

Preprocessing:

1. Compute the rolling average of the force (Otherwise the gradient would be too wild)
2. Compute the gradient (Converting to gradient space gets rid of linear contributions in the approach part)
3. Compute the rolling average of the gradient (Makes the curve to analyze more smooth so that the methods below don't hit the alarm too early)

Method 1: baseline deviation

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

Method 2: sign of gradient

1. Apply a median filter to 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, then a combined constant+linear function ( $\max(\text{constant}, \text{linear})$ ) is fitted to the gradient to determine the contact point. If that fails as well, then the CP defaults to the center of the entire approach curve.

Changed in version 1.6.0: Add the gradient preprocessing step to circumvent issues with tilted baselines. This feature does not significantly affect fitting results.

Changed in version 1.6.1: Added  $\max(\text{constant}, \text{linear})$  fit when the other methods fail.

### `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*, *fmt*='tab')

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\_ancillary\_parameters** (*model\_key=None*)

Compute ancillary parameters for the current model

**get\_initial\_fit\_parameters** (*model\_key=None*, *common\_ancillaries=True*,  
*model\_ancillaries=True*)

Return the initial fit parameters

If there are not initial fit parameters set in *self.fit\_properties*, then they are computed.

#### Parameters

- **model\_key** (*str*) – Optionally set a model key. This will override the “model\_key” key in *self.fit\_properties*.
- **common\_ancillaries** (*bool*) – Guess global ancillaries such as the contact point.
- **model\_ancillaries** (*bool*) – Guess model-related ancillaries

#### Notes

*global\_ancillaries* and *model\_ancillaries* only have an effect if *self.fit\_properties*["params\_initial"] is set.

**get\_rating\_parameters** ()

Return current rating 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**

All data as `afmformats.AFMForceDistance`

**property fit\_properties**

Fitting results, see *Indentation.fit\_model()*

**preprocessing**

Default preprocessing steps, see *Indentation.apply\_preprocessing()*.

## 7.3 Groups

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

Group of Indentation

**Parameters**

- **path** (*str or pathlib.Path or None*) – 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*)

Append an indentation dataset

**Parameters** **item** (`nanite.indent.Indentation`) – Force-indentation dataset

**get\_enum** (*enum*)

Return the indentation curve with this enum value

**Raises**

- **ValueError** if multiple curves with the same enum value exist. –
- **KeyError** if the enum value is not found –

**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-distance data

**Return type** `nanite.IndentationGroup`

## 7.4 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 extension, 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 `afmformats.AFMForceDistance`

## 7.5 Preprocessing

**exception** `nanite.preproc.CannotSplitWarning`

**class** `nanite.preproc.IndentationPreprocessor`

**static apply** (*apret, preproc\_names*)

Perform force-distance 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
```

## 7.6 Modeling

### 7.6.1 Methods and constants

**exception nanite.model.ModelImplementationError**

**exception nanite.model.ModelImplementationWarning**

**exception nanite.model.ModelIncompleteError**

**nanite.model.get\_anc\_parm\_keys (model\_key)**

Return the key names of a model’s ancillary parameters

**nanite.model.get\_anc\_parms (idnt, model\_key)**

Compute ancillary parameters for a force-distance dataset

Ancillary parameters include parameters that:

- are unrelated to fitting: They may just be important parameters to the user.

- require the entire dataset: They cannot be extracted during fitting, because they require more than just the approach xor retract curve to compute (e.g. hysteresis, jump of retract curve at maximum indentation). They may, additionally, depend on initial fit parameters set by the user.
- require a fit: They are dependent on fitting parameters but are not required during fitting.

## Notes

If an ancillary parameter name matches that of a fitting parameter, then it is assumed that it can be used for fitting. Please see `nanite.indent.Indentation.get_initial_fit_parameters()` and `nanite.fit.guess_initial_parameters()`.

Ancillary parameters are set to `np.nan` if they cannot be computed.

### Parameters

- **idnt** (`nanite.indent.Indentation`) – The force-distance data for which to compute the ancillary parameters
- **model\_key** (`str`) – Name of the model

**Returns** `ancillaries` – key-value dictionary of ancillary parameters

**Return type** `collections.OrderedDict`

`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)`

Return 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 label (e.g. “Young’s Modulus”)

**Return type** `str`

`nanite.model.get_parm_unit(model_key, parm_key)`

Return parameter unit

### Parameters

- **model\_key** (`str`) – The model key (e.g. “hertz\_cone”)
- **parm\_key** (`str`) – The parameter key (e.g. “E”)

**Returns** `parm_unit` – The parameter unit (e.g. “Pa”)

**Return type** `str`

`nanite.model.register_model(module, module_name)`

Register a fitting model

`nanite.model.ANCILLARY_COMMON = {'max_indent': ('Maximum indentation', 'm')}`

Common ancillary parameters

## 7.6.2 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(delta, E, 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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **alpha** (*float*) – Half cone angle [degrees]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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]

`nanite.model.model_conical_indenter.model_func` (*delta*, *E*, *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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **alpha** (*float*) – Half cone angle [degrees]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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_paraboloidal

nanite.model.model\_hertz\_paraboloidal.**hertz\_paraboloidal**(*delta*, *E*, *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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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]

`nanite.model.model_hertz_paraboloidal.model_func` (*delta*, *E*, *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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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**(*delta*,  
*E*,  
*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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **alpha** (*float*) – Inclination angle of the pyramidal face [degrees]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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.
- The inclination angle of the pyramidal face (in radians) must be close to zero.

## References

Bilodeau et al. 1992 [Bil92]

`nanite.model.model_hertz_three_sided_pyramid.model_func` (*delta*, *E*, *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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **alpha** (*float*) – Inclination angle of the pyramidal face [degrees]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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.
- The inclination angle of the pyramidal face (in radians) must be close to zero.

## 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` (*a*, *R*)  
 Compute indentation from contact area radius (wrapper)

`nanite.model.model_sneddon_spherical.get_a` (*R*, *delta*, *accuracy=1e-12*)  
 Compute the contact area radius (wrapper)

nanite.model.model\_sneddon\_spherical.**hertz\_spherical** (*delta*, *E*, *R*, *nu*, *contact\_point=0.0*, *baseline=0.0*)

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

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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](#)]

nanite.model.model\_sneddon\_spherical.**model\_func** (*delta*, *E*, *R*, *nu*, *contact\_point=0.0*, *baseline=0.0*)

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**

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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** (*delta*,  
*E*,  
*R*,  
*nu*,  
*con-*  
*tact\_point=0*,  
*base-*  
*line=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**

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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], Dobler (personal communication, 2018) [Dob18]

`nanite.model.model_sneddon_spherical_approximation.model_func(delta, E, 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**

- **delta** (*1d ndarray*) – Indentation [m]
- **E** (*float*) – Young’s modulus [N/m<sup>2</sup>]
- **R** (*float*) – Tip radius [m]
- **nu** (*float*) – Poisson’s ratio
- **contact\_point** (*float*) – Indentation offset [m]
- **baseline** (*float*) – Force offset [N]

**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], Dobler (personal communication, 2018) [Dob18]

## 7.7 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` (*idnt*, **\*\*kwargs**)

Fit force-distance curves

### Parameters

- **idnt** (`nanite.indent.Indentation`) – The dataset to fit
- **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** (*idnt=None, model\_key='hertz\_para'*)  
Get initial fit parameters for a specific model

`nanite.fit.guess_initial_parameters` (*idnt=None, model\_key='hertz\_para', common\_ancillaries=True, model\_ancillaries=True*)  
Guess initial fitting parameters

#### Parameters

- **idnt** (`nanite.indent.Indentation`) – The dataset to use for guessing initial fitting parameters using ancillary parameters
- **model\_key** (*str*) – The model key
- **common\_ancillaries** (*bool*) – Guess global ancillary parameters (such as contact point)
- **model\_ancillaries** (*bool*) – Use model-related ancillary parameters

`nanite.fit.obj2bytes` (*obj*)  
Bytes representation of an object for hashing



## 7.8 Rating

### 7.8.1 Features

**class** `nanite.rate.features.IndentationFeatures` (*dataset=None*)

**static compute\_features** (*idnt, which\_type='all', names=None, ret\_names=False*)  
 Compute the features for a data set

#### Parameters

- **idnt** (`nanite.Indentation`) – A dataset to rate
- **names** (*list of str*) – The names of the rating methods to use, e.g. ["rate\_apr\_bumps", "rate\_apr\_mon\_incr"]. If None (default), all available rating methods are used.

#### Notes

*names* may include features that are excluded by *which\_type*. E.g. if a “bool” feature is in *names* but *which\_type* is “float”, then the “bool” feature will be silently ignored.

**feat\_bin\_apr\_spikes\_count** ()  
 spikes during IDT

Sudden spikes in indentation curve

**feat\_bin\_cp\_position** ()  
 CP outside of data range

Contact point position outside of range

**feat\_bin\_size** ()  
 dataset too small

Number of points in indentation curve

**feat\_con\_apr\_flatness** ()  
 flatness of APR residuals

fraction of the positive-gradient residuals in the approach part

**feat\_con\_apr\_size** ()  
 relative APR size

length of the approach part relative to the indentation part

**feat\_con\_apr\_sum** ()  
 residuals of APR

absolute sum of the residuals in the approach part

**feat\_con\_bln\_slope** ()  
 slope of BLN

slope obtained from a linear least-squares fit to the baseline

**feat\_con\_bln\_variation** ()  
 variation in BLN

comparison of the forces at the beginning and at the end of the baseline

**feat\_con\_cp\_curvature** ()  
curvature at CP

curvature of the force-distance data at the contact point

**feat\_con\_cp\_magnitude** ()  
residuals at CP

mean value of the residuals around the contact point

**feat\_con\_idt\_maxima\_75perc** ()  
maxima in IDT residuals

sum of the indentation residuals' maxima in three intervals in-between 25% and 100% relative to the maximum indentation

**feat\_con\_idt\_monotony** ()  
monotony of IDT

change of the gradient in the indentation part

**feat\_con\_idt\_spike\_area** ()  
area of IDT spikes

area of spikes appearing in the indentation part

**feat\_con\_idt\_sum** ()  
overall IDT residuals

sum of the residuals in the indentation part

**feat\_con\_idt\_sum\_75perc** ()  
residuals in 75% IDT

sum of the residuals in the indentation part in-between 25% and 100% relative to the maximum indentation

**classmethod get\_feature\_funcs** (*which\_type='all', names=None*)  
Return functions that compute features from a dataset

#### Parameters

- **names** (*list of str*) – The names of the rating methods to use, e.g. ["rate\_apr\_bumps", "rate\_apr\_mon\_incr"]. If None (default), all available rating methods are returned.
- **which\_type** (*str*) – Which features to return: ["all", "bool", "float"].

**Returns** **raters** – Each item in the list consists contains the name of the rating method and the corresponding rating method.

**Return type** list of tuples (name, callable)

**classmethod get\_feature\_names** (*which\_type='all', names=None, ret\_indices=False*)  
Get features names

#### Parameters

- **which\_type** (*str or list of str*) – Return only features that are of a certain type. See `VALID_FEATURE_TYPES` for valid strings.
- **names** (*list of str*) – Return only features that are in this list.
- **ret\_indices** (*bool*) – If True, also return the internal feature indices.

**Returns** **name\_list** – List of feature names (callables of this class)

**Return type** list of str

```

property contact_point
property datafit_apr
property datares_apr
dataset
    current dataset from which features are computed
property datax_apr
property datay_apr
property has_contact_point
property is_fitted
property is_valid
property meta

```

```

nanite.rate.features.VALID_FEATURE_TYPES = ['all', 'binary', 'continuous']
    Valid keyword arguments for feature types

```

## 7.8.2 Rater

```

class nanite.rate.rater.IndentationRater (regressor=None, scale=None, lda=None, training_set=None, names=None, weight=True, sample_weight=None, *args, **kwargs)

```

Rate quality

### Parameters

- **regressor** (*scikit-learn RegressorMixin*) – The regressor used for rating
- **scale** (*bool*) – If True, apply a Standard Scaler. If a regressor based on decision trees is used, the Standard Scaler is not used by default, otherwise it is.
- **lda** (*bool*) – If True, apply a Linear Discriminant Analysis (LDA). If a regressor based on a decision tree is used, LDA is not used by default, otherwise it is.
- **training\_set** (*tuple of (X, y)*) – The training set (samples, response)
- **names** (*list of str*) – Feature names to use
- **weight** (*bool*) – Weight the input samples by the number of occurrences or with *sample\_weight*. For tree-based classifiers, set this to True to avoid bias.
- **sample\_weight** (*list-like*) – The sample weights. If set to *None* sample weights are computed from the training set.
- **\*args** (*list*) – Positional arguments for IndentationFeatures
- **\*\*kwargs** – Keyword arguments for IndentationFeatures

See also:

[sklearn.preprocessing.StandardScaler](#) Standard scaler

[sklearn.discriminant\\_analysis.LinearDiscriminantAnalysis](#) Linear discriminant analysis

[nanite.rate.regressors.reg\\_trees](#) List of regressors that are identified as tree-based

**static compute\_sample\_weight** (*X*, *y*)

Weight samples according to occurrence in *y*

**static get\_training\_set\_path** (*label*='zef18')

Return the path to a training set shipped with nanite

Training sets are stored in the *nanite.rate* module path with `ts_` prepended to *label*.

**classmethod load\_training\_set** (*path*=None, *names*=None, *which\_type*=['continuous'], *remove\_nan*=True, *ret\_names*=False)

Load a training set from a directory

By default, only the “continuous” features are imported. The “binary” features are not needed for training; they are used to sort out new force-distance data.

**rate** (*samples*=None, *datasets*=None)

Perform rating step

#### Parameters

- **samples** (*1d or 2d ndarray (cast to 2d ndarray) or None*) – Measured samples, if set to None, *dataset* must be given.
- **dataset** (*list of nanite.Indentation*) – Full, fitted measurement

**Returns ratings** – Resulting ratings

**Return type** *list*

**names**

feature names used by the regressor pipeline

**pipeline**

sklearn pipeline with transforms (and regressor if given)

*nanite.rate.rater.get\_available\_training\_sets* ()

List of internal training sets

*nanite.rate.rater.get\_rater* (*regressor*, *training\_set*='zef18', *names*=None, *lda*=None, *\*\*reg\_kwargs*)

Convenience method to get a rater

#### Parameters

- **regressor** (*str or RegressorMixin*) – If a string, must be in *reg\_names*.
- **training\_set** (*str or pathlib.Path or tuple (X, y)*) – A string label representing a training set shipped with nanite, the path to a training set, or a tuple representing the training set (samples, response) for use with sklearn.

**Returns irater** – The rating instance.

**Return type** *nanite.IndentationRater*

### 7.8.3 Regressors

scikit-learn regressors and their keyword arguments

`nanite.rate.regressors.reg_names = ['AdaBoost', 'Decision Tree', 'Extra Trees', 'Gradient Boosting']`  
 List of available default regressor names

`nanite.rate.regressors.reg_trees = ['AdaBoostRegressor', 'DecisionTreeRegressor', 'ExtraTreeRegressor']`  
 List of tree-based regressor class names (used for keyword defaults in `IndentationRater`)

### 7.8.4 Manager

Save and load user-rated datasets

**class** `nanite.rate.io.RateManager` (*path*, *verbose=0*)  
 Manage user-defined rates

**export\_training\_set** (*path*)

**get\_cross\_validation\_score** (*regressor*, *training\_set=None*, *n\_splits=20*, *random\_state=42*)  
 Regressor cross-validation scoring

Cross-validation is used to identify regressors that over-fit the train set by splitting the train set into multiple learn/test sets and quantifying the regressor performance for each split.

#### Parameters

- **regressor** (*str* or *RegressorMixin*) – If a string, must be in *reg\_names*.
- **training\_set** (*X*, *y*) – If given, do not use `self.samples`

#### Notes

A `skimage.model_selection.KFold` cross validator is used in combination with the mean squared error score.

Cross-validation score is computed from samples that are filtered with the binary features and only from samples that do not contain any nan values.

**get\_rates** (*which='user'*, *training\_set='zef18'*)

**which:** **str** Which rating to return: “user” or a regressor name

**get\_training\_set** (*which\_type='all'*, *prefilter\_binary=False*, *remove\_nans=False*, *transform=False*)

Return (*X*, *y*) training set

**property datasets**

**path**

Path to the manual ratings (directory or .h5 file)

**property ratings**

**property samples**

The individual sample ratings computed by `afmlib`

**verbose**

verbosity level

`nanite.rate.io.hash_file` (*path*, *blocksize=65536*)  
 Compute sha256 hex-hash of a file

**Parameters**

- **path** (*str* or *pathlib.Path*) – path to the file
- **blocksize** (*int*) – block size read from the file

**Returns** `hex` – The first six characters of the hash

**Return type** `str`

`nanite.rate.io.hdf5_rated(h5path, indent)`

Test whether an indentation has already been rated

**Returns**

**Return type** `is_rated, rating, comment`

`nanite.rate.io.load(path, meta_only=False, verbose=0)`

**Notes**

The `.fit_properties` attribute of each `Indentation` instance is overridden by a simple dictionary, so its functionalities are not available anymore.

`nanite.rate.io.load_hdf5(path, meta_only=False)`

`nanite.rate.io.save_hdf5(h5path, indent, user_rate, user_name, user_comment, h5mode='a')`

Store all relevant data of a user rating into an hdf5 file

**Parameters**

- **h5path** (*str*) – Path to HDF5 rating container where data will be stored
- **indent** (`nanite.Indentation`) – The experimental data processed and fitted with nanite
- **user\_rate** (*float*) – Rating given by the user
- **user\_name** (*str*) – Name of the rating user

## 7.9 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(indent)`

`feat_fit_contact_point(indent)`

`feat_fit_youngs_modulus(indent)`

`feat_meta_rating(indent)`

**feat\_meta\_scan\_order** (*idnt*)

**get\_coords** (*which='px'*)

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

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

**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

**property extent**

extent (x1, x2, y1, y2) [ $\mu\text{m}$ ]

**features**

Available features (see *nanite.qmap.available\_features*)

**group**

Indentation data (instance of *nanite.IndentationGroup*)

**property shape**

shape of the map [px]

`nanite.qmap.available_features = ['data min height', 'fit contact point', "fit young's mod"]`  
 Available features for quantitative maps





## CHANGELOG

List of changes in-between nanite releases.

### 8.1 version 1.7.0

- enh: simplified writing new model functions by introducing default modeling and residual wrappers
- ref: improve code readability

### 8.2 version 1.6.3

- tests: fix fails due to tiff file upgrade
- setup: lift historic pinning of lmfit==0.9.5

### 8.3 version 1.6.2

- tests: improve coverage
- enh: add sanity checks for models during registration (#5)

### 8.4 version 1.6.1

- enh: if the contact point estimate is not possible, use a fit with a partially constant and linear function

### 8.5 version 1.6.0

- enh: improve contact point estimation by computing the gradient first; resolves issues with tilted baselines (#6)  
(This may affect fitting results slightly, hence the new minor release)

## 8.6 version 1.5.5

- setup: make tkinter optional for frozen applications

## 8.7 version 1.5.4

- setup: bump scikit-learn from 0.18.0 to 0.23.0 (different model results due to bugfixes, enhancements, or random sampling procedures; the tests have been updated accordingly)
- setup: bump afmformats from 0.10.0 to 0.10.2

## 8.8 version 1.5.3

- setup: new builds for Python 3.8

## 8.9 version 1.5.2

- enh: be more verbose when tip position cannot be computed
- setup: bump afmformats from 0.7.0 to 0.10.0

## 8.10 version 1.5.1

- setup: bump afmformats from 0.6.0 to 0.7.0 (metadata fixes)

## 8.11 version 1.5.0

- feat: IndentationGroup.get\_enum returns a curve from an enum value
- setup: bump afmformats from 0.5.0 to 0.6.0 (hdf5 export, improved tab export)

## 8.12 version 1.4.1

- enh: set parameter *baseline* to “vary” for all models
- fix: make sure that *model\_key* is set before *params\_initial* when fitting with kwargs (otherwise, *params\_initial* might reset)

## 8.13 version 1.4.0

- feat: add function *Indentation.get\_rating\_parameters*
- feat: compute additional ancillary parameter “Maximum indentation”
- feat: new functions *model.get\_parm\_unit* and updated *model.get\_parm\_name* to work with ancillary parameters as well

## 8.14 version 1.3.0

- feat: allow to define ancillary parameters for models and use them during fitting by default
- feat: *Indentation.get\_initial\_fit\_parameters* now automatically computes common and model-related ancillary parameters if no initial parameters are present
- enh: allow to set the *model\_key* in more functions of *Indentation*
- ref: use *idnt* to represent *Indentation* instances
- fix: preprocessing steps not stored in *Indentation.preprocessing*
- setup: bump afmformats from 0.4.1 to 0.5.0

## 8.15 version 1.2.4

- enh: update boundaries and default values for model parameters

## 8.16 version 1.2.3

- fix: *FitProperties* did not detect changes in “params\_initial”

## 8.17 version 1.2.2

- setup: bump afmformats version from 0.3.0 to 0.4.1

## 8.18 version 1.2.1

- enh: skip computation of tip position if it is already in the dataset and cannot be computed e.g. due to missing spring constant
- fix: typo in *get\_data\_paths\_enum*
- setup: bump afmformats version from 0.2.0 to 0.3.0

## 8.19 version 1.2.0

- tests: np.asscalar is deprecated
- ref: migrate to afmformats (#1)
- docs: minor improvements

## 8.20 version 1.1.2

- fix: add `__version__` property
- tests: use `time.perf_counter` for timing tests
- docs: improved LaTeX rendering

## 8.21 version 1.1.1

- setup: migrate to PEP 517 (pyproject.toml)
- docs: minor update

## 8.22 version 1.1.0

- feat: add contact point to available features in qmap visualization
- fix: avoid two invalid operations when computing features

## 8.23 version 1.0.1

- fix: invalid operation when loading data with a callback function

## 8.24 version 1.0.0

- docs: minor update

## 8.25 version 0.9.3

- enh: store nanite and h5py library versions in rating container
- enh: update hyperparameters of rating regressors
- ref: deprecation in h5py: replace `dataset.value` by `dataset[...]`

## 8.26 version 0.9.2

- ref: renamed the mode *model\_hertz\_parabolic* to *model\_hertz\_paraboloidal* to be consistent
- docs: updat code reference and other minor improvements

## 8.27 version 0.9.1

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

## 8.28 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

## 8.29 version 0.8.0

- initial release



**BILBLIOGRAPHY**





## 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.
- [Dob18] Wolfgang Dobler. Truncated power series approximation for the relationship between indentation and force of a spherical indenter in atomic force microscopy. personal communication with Wolfgang Dobler, JPK Instruments, Berlin, November 2018.
- [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.
- [MKH+19] Stephanie Möllmert, Maria A. Kharlamova, Tobias Hoche, Anna V. Taubenberger, Shada Abuhattum, Veronika Kuscha, Michael Brand, and Jochen Guck. Zebrafish spinal cord repair is accompanied by transient tissue stiffening. (*manuscript in preparation*), 2019.
- [MAM+19] Paul Müller, Shada Abuhattum, Stephanie Möllmert, Elke Ulbricht, Anna V. Taubenberger, and Jochen Guck. Nanite: using machine learning to assess the quality of atomic force microscopy-enabled nano-indentation data. *BMC Bioinformatics*, 20(1):1–9, sep 2019. doi:10.1186/s12859-019-3010-3.
- [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

- nanite.fit, 43
- nanite.group, 30
- nanite.indent, 27
- nanite.model, 32
  - nanite.model.model\_conical\_indenter, 34
  - nanite.model.model\_hertz\_paraboloidal, 36
  - nanite.model.model\_hertz\_three\_sided\_pyramid, 38
  - nanite.model.model\_sneddon\_spherical, 39
  - nanite.model.model\_sneddon\_spherical\_approximation, 41
- nanite.preproc, 31
- nanite.qmap, 50
- nanite.rate.features, 45
- nanite.rate.io, 49
- nanite.rate.rater, 47
- nanite.rate.regressors, 49
- nanite.read, 31



## A

ANCILLARY\_COMMON (in module *nanite.model*), 33  
 append() (*nanite.group.IndentationGroup* method), 30  
 apply() (*nanite.preproc.IndentationPreprocessor* static method), 31  
 apply\_preprocessing() (*nanite.indent.Indentation* method), 27  
 available() (*nanite.preproc.IndentationPreprocessor* static method), 32  
 available\_features (in module *nanite.qmap*), 51  
 available\_preprocessors (in module *nanite.preproc*), 32

## B

built-in function  
 nanite.load\_group(), 27

## C

CannotSplitWarning, 31  
 compute\_emodulus\_mindelta() (*nanite.indent.Indentation* method), 27  
 compute\_emodulus\_vs\_mindelta() (*nanite.fit.IndentationFitter* method), 44  
 compute\_features() (*nanite.rate.features.IndentationFeatures* static method), 45  
 compute\_opt\_mindelta() (*nanite.fit.IndentationFitter* static method), 44  
 compute\_sample\_weight() (*nanite.rate.rater.IndentationRater* static method), 47  
 compute\_tip\_position() (*nanite.preproc.IndentationPreprocessor* static method), 32  
 contact\_point() (*nanite.rate.features.IndentationFeatures* property), 46  
 correct\_force\_offset() (*nanite.preproc.IndentationPreprocessor* static method), 32  
 correct\_split\_approach\_retract() (*nanite.preproc.IndentationPreprocessor*

static method), 32  
 correct\_tip\_offset() (*nanite.preproc.IndentationPreprocessor* static method), 32

## D

data (*nanite.indent.Indentation* attribute), 30  
 datafit\_apr() (*nanite.rate.features.IndentationFeatures* property), 47  
 DataMissingWarning, 50  
 datares\_apr() (*nanite.rate.features.IndentationFeatures* property), 47  
 dataset (*nanite.rate.features.IndentationFeatures* attribute), 47  
 datasets() (*nanite.rate.io.RateManager* property), 49  
 datax\_apr() (*nanite.rate.features.IndentationFeatures* property), 47  
 datay\_apr() (*nanite.rate.features.IndentationFeatures* property), 47  
 delta\_of\_a() (in module *nanite.model.model\_sneddon\_spherical*), 39

## E

estimate\_contact\_point\_index() (*nanite.indent.Indentation* method), 28  
 estimate\_optimal\_mindelta() (*nanite.indent.Indentation* method), 28  
 export() (*nanite.indent.Indentation* method), 28  
 export\_training\_set() (*nanite.rate.io.RateManager* method), 49  
 extent() (*nanite.qmap.QMap* property), 51

## F

feat\_bin\_apr\_spikes\_count() (*nanite.rate.features.IndentationFeatures* method), 45  
 feat\_bin\_cp\_position() (*nanite.rate.features.IndentationFeatures* method), 45  
 feat\_bin\_size() (*nanite.rate.features.IndentationFeatures* method), 45

- `feat_con_apr_flatness()`  
*(nanite.rate.features.IndentationFeatures method)*, 45
- `feat_con_apr_size()`  
*(nanite.rate.features.IndentationFeatures method)*, 45
- `feat_con_apr_sum()`  
*(nanite.rate.features.IndentationFeatures method)*, 45
- `feat_con_bln_slope()`  
*(nanite.rate.features.IndentationFeatures method)*, 45
- `feat_con_bln_variation()`  
*(nanite.rate.features.IndentationFeatures method)*, 45
- `feat_con_cp_curvature()`  
*(nanite.rate.features.IndentationFeatures method)*, 45
- `feat_con_cp_magnitude()`  
*(nanite.rate.features.IndentationFeatures method)*, 46
- `feat_con_idt_maxima_75perc()`  
*(nanite.rate.features.IndentationFeatures method)*, 46
- `feat_con_idt_monotony()`  
*(nanite.rate.features.IndentationFeatures method)*, 46
- `feat_con_idt_spike_area()`  
*(nanite.rate.features.IndentationFeatures method)*, 46
- `feat_con_idt_sum()`  
*(nanite.rate.features.IndentationFeatures method)*, 46
- `feat_con_idt_sum_75perc()`  
*(nanite.rate.features.IndentationFeatures method)*, 46
- `feat_data_min_height_measured_um()`  
*(nanite.qmap.QMap method)*, 50
- `feat_fit_contact_point()` *(nanite.qmap.QMap method)*, 50
- `feat_fit_youngs_modulus()`  
*(nanite.qmap.QMap method)*, 50
- `feat_meta_rating()` *(nanite.qmap.QMap method)*, 50
- `feat_meta_scan_order()` *(nanite.qmap.QMap method)*, 50
- `features` *(nanite.qmap.QMap attribute)*, 51
- `fit()` *(nanite.fit.IndentationFitter method)*, 44
- `fit_model()` *(nanite.indent.Indentation method)*, 29
- `fit_properties()` *(nanite.indent.Indentation property)*, 30
- `FitDataError`, 43
- `FitKeyError`, 43
- `FitProperties` *(class in nanite.fit)*, 43
- `FitWarning`, 43
- ## G
- `get_a()` *(in module nanite.model.model\_sneddon\_spherical)*, 39
- `get_anc_parm_keys()` *(in module nanite.model)*, 32
- `get_anc_parms()` *(in module nanite.model)*, 32
- `get_ancillary_parameters()`  
*(nanite.indent.Indentation method)*, 29
- `get_available_training_sets()` *(in module nanite.rate.rater)*, 48
- `get_coords()` *(nanite.qmap.QMap method)*, 51
- `get_cross_validation_score()`  
*(nanite.rate.io.RateManager method)*, 49
- `get_data_paths()` *(in module nanite.read)*, 31
- `get_data_paths_enum()` *(in module nanite.read)*, 31
- `get_enum()` *(nanite.group.IndentationGroup method)*, 30
- `get_feature_funcs()`  
*(nanite.rate.features.IndentationFeatures class method)*, 46
- `get_feature_names()`  
*(nanite.rate.features.IndentationFeatures class method)*, 46
- `get_init_parms()` *(in module nanite.model)*, 33
- `get_initial_fit_parameters()`  
*(nanite.indent.Indentation method)*, 29
- `get_initial_parameters()`  
*(nanite.fit.IndentationFitter method)*, 44
- `get_model_by_name()` *(in module nanite.model)*, 33
- `get_parm_name()` *(in module nanite.model)*, 33
- `get_parm_unit()` *(in module nanite.model)*, 33
- `get_qmap()` *(nanite.qmap.QMap method)*, 51
- `get_rater()` *(in module nanite.rate.rater)*, 48
- `get_rates()` *(nanite.rate.io.RateManager method)*, 49
- `get_rating_parameters()`  
*(nanite.indent.Indentation method)*, 29
- `get_training_set()` *(nanite.rate.io.RateManager method)*, 49
- `get_training_set_path()`  
*(nanite.rate.rater.IndentationRater static method)*, 48
- `group` *(nanite.qmap.QMap attribute)*, 51
- `guess_initial_parameters()` *(in module nanite.fit)*, 44
- ## H
- `has_contact_point()`  
*(nanite.rate.features.IndentationFeatures*



- property*), 47
- hash\_file() (in module *nanite.rate.io*), 49
- hdf5\_rated() (in module *nanite.rate.io*), 50
- hertz\_conical() (in module *nanite.model.model\_conical\_indenter*), 34
- hertz\_paraboloidal() (in module *nanite.model.model\_hertz\_paraboloidal*), 36
- hertz\_sneddon\_spherical\_approx() (in module *nanite.model.model\_sneddon\_spherical\_approximation*), 41
- hertz\_spherical() (in module *nanite.model.model\_sneddon\_spherical*), 40
- hertz\_three\_sided\_pyramid() (in module *nanite.model.model\_hertz\_three\_sided\_pyramid*), 38
- ## I
- Indentation (class in *nanite.indent*), 27
- IndentationFeatures (class in *nanite.rate.features*), 45
- IndentationFitter (class in *nanite.fit*), 43
- IndentationGroup (class in *nanite.group*), 30
- IndentationPreprocessor (class in *nanite.preproc*), 31
- IndentationRater (class in *nanite.rate.rater*), 47
- index() (*nanite.group.IndentationGroup* method), 30
- is\_fitted() (*nanite.rate.features.IndentationFeatures* property), 47
- is\_valid() (*nanite.rate.features.IndentationFeatures* property), 47
- ## L
- load() (in module *nanite.rate.io*), 50
- load\_data() (in module *nanite.read*), 31
- load\_group() (in module *nanite.group*), 30
- load\_hdf5() (in module *nanite.rate.io*), 50
- load\_training\_set() (*nanite.rate.rater.IndentationRater* class method), 48
- ## M
- meta() (*nanite.rate.features.IndentationFeatures* property), 47
- model\_doc (*nanite.model.nanite.model.model\_submodule* attribute), 34
- model\_func() (in module *nanite.model.model\_conical\_indenter*), 35
- model\_func() (in module *nanite.model.model\_hertz\_paraboloidal*), 37
- model\_func() (in module *nanite.model.model\_hertz\_three\_sided\_pyramid*), 39
- model\_func() (in module *nanite.model.model\_sneddon\_spherical*), 40
- model\_func() (in module *nanite.model.model\_sneddon\_spherical\_approximation*), 42
- model\_key (*nanite.model.nanite.model.model\_submodule* attribute), 34
- model\_name (*nanite.model.nanite.model.model\_submodule* attribute), 34
- ModelImplementationError, 32
- ModelImplementationWarning, 32
- ModelIncompleteError, 32
- module
- nanite.fit*, 43
  - nanite.group*, 30
  - nanite.indent*, 27
  - nanite.model*, 32
  - nanite.model.model\_conical\_indenter*, 34
  - nanite.model.model\_hertz\_paraboloidal*, 36
  - nanite.model.model\_hertz\_three\_sided\_pyramid*, 38
  - nanite.model.model\_sneddon\_spherical*, 39
  - nanite.model.model\_sneddon\_spherical\_approximation*, 41
  - nanite.preproc*, 31
  - nanite.qmap*, 50
  - nanite.rate.features*, 45
  - nanite.rate.io*, 49
  - nanite.rate.rater*, 47
  - nanite.rate.regressors*, 49
  - nanite.read*, 31
- ## N
- names (*nanite.rate.rater.IndentationRater* attribute), 48
- nanite.fit*
- module, 43
- nanite.group*
- module, 30
- nanite.indent*
- module, 27
- nanite.Indentation* (built-in class), 27
- nanite.IndentationGroup* (built-in class), 27
- nanite.IndentationRater* (built-in class), 27
- nanite.load\_group*()
- built-in function, 27
- nanite.model*
- module, 32

nanite.model.model\_conical\_indenter module, 34

nanite.model.model\_hertz\_paraboloidal module, 36

nanite.model.model\_hertz\_three\_sided\_pyramid module, 38

nanite.model.model\_sneddon\_spherical module, 39

nanite.model.model\_sneddon\_spherical\_application module, 41

nanite.model.model\_submodule.get\_parameter\_defaults() (in module nanite.model), 34

nanite.model.model\_submodule.model() (in module nanite.model), 34

nanite.model.model\_submodule.residual() (in module nanite.model), 34

nanite.preproc module, 31

nanite.qmap module, 50

nanite.QMap (built-in class), 27

nanite.rate.features module, 45

nanite.rate.io module, 49

nanite.rate.rater module, 47

nanite.rate.regressors module, 49

nanite.read module, 31

rate\_quality() (nanite.indent.Indentation method), 29

RateManager (class in nanite.rate.io), 49

ratings() (nanite.rate.io.RateManager property), 49

reg\_names (in module nanite.rate.regressors), 49

reg\_trees (in module nanite.rate.regressors), 49

register\_model() (in module nanite.model), 33

reset() (nanite.fit.FitProperties method), 43

reset() (nanite.indent.Indentation method), 30

restore() (nanite.fit.FitProperties method), 43

restore\_defaults()

## S

samples() (nanite.rate.io.RateManager property), 49

save\_hdf5() (in module nanite.rate.io), 50

shape() (nanite.qmap.QMap property), 51

smooth\_height() (nanite.preproc.IndentationPreprocessor static method), 32

subgroup\_with\_path() (nanite.group.IndentationGroup method), 30

## V

VALID\_FEATURE\_TYPES (in module nanite.rate.features), 47

verbose (nanite.rate.io.RateManager attribute), 49

## O

obj2bytes() (in module nanite.fit), 44

## P

parameter\_keys (nanite.model.nanite.model.model\_submodule attribute), 34

parameter\_names (nanite.model.nanite.model.model\_submodule attribute), 34

parameter\_units (nanite.model.nanite.model.model\_submodule attribute), 34

path (nanite.rate.io.RateManager attribute), 49

pipeline (nanite.rate.rater.IndentationRater attribute), 48

preprocessing (nanite.indent.Indentation attribute), 30

## Q

QMap (class in nanite.qmap), 50

## R

rate() (nanite.rate.rater.IndentationRater method), 48