stratagemtools Documentation

Release 0+untagged.74.g38c8081.dirty

Philippe Pinard

May 06, 2016

Contents

Warning: Requires a legal version of SAMx's STRATAGem installed on a Windows PC to work, including the USB dongle.

stratagemtools is a Python interface to SAMx's STRATAGem program. The interface allows, for example, to calculate k-ratios and compute film thickness in Python using the STRATAGem's OEM interface, i.e. without the normal graphical interface. It gives an object oriented approach to microanalysis calculations.

stratagemtools was developed as part of the doctorate thesis project of Philippe T. Pinard at RWTH Aachen University (Aachen, Germany) under the supervision of Dr. Silvia Richter.

Contents:

CHAPTER 1

Installation guide

1.1 Requirements

stratagemtools requires:

- Microsoft Windows
- STRATAGem >= 4.1.0, stratadll.dll >= 4.7.0
- Python 3.x (32-bit, does not work with 64-bit)

and the following Python libraires, which should be automatically installed by pip.

- pyparsing
- nose (for testing only)

1.2 Installation of release version

Run in a command prompt:

```
py -3 -m pip install stratagemtools
```

1.3 Installation of developer version

Run in a command prompt:

```
git clone https://github.com/ppinard/stratagemtools.git
cd stratagemtools
py -3 -m pip install -e .
```

CHAPTER 2

API reference

The following gives the definitions of all classes, methods and functions of *stratagemtools*.

Contents:

2.1 Element properties

Various properties of atoms.

stratagemtools.element_properties.**symbol**(*z*)
Returns the element's symbol.

Parameters **z** (int) – atomic number

Returns symbol

Return type str

stratagemtools.element_properties.**atomic_number**(*symbol*)
Returns the atomic number for the specified *symbol*. This function is case insensitive.

Parameters **symbol** (str) – symbol of the element (e.g. C)

Returns atomic number

Return type int

stratagemtools.element_properties.**mass_density_kg_m3**(*z*)
Returns the mass density in kilograms per cubic meter for the specified atomic number.

Parameters **z** (int) – atomic number

Returns mass density

Return type float

stratagemtools.element_properties.**atomic_mass_kg_mol**(*z*)
Returns the atomic molar mass in kilograms per mol for the specified atomic number.

Parameters **z** (int) – atomic number

Returns atomic molar mass

Return type float

2.2 Experiment

Definition of an experiment, structure to setup experimental parameters and measurements in STRATAGem.

`stratagemtools.experiment.`**LINE_KA = 0**
X-ray line Kα

`stratagemtools.experiment.`**LINE_KB = 1**
X-ray line Kβ

`stratagemtools.experiment.`**LINE_LA = 2**
X-ray line Lα

`stratagemtools.experiment.`**LINE_LB = 3**
X-ray line Lβ

`stratagemtools.experiment.`**LINE_MA = 4**
X-ray line Mα

`stratagemtools.experiment.`**LINE_MB = 5**
X-ray line Mβ

class `stratagemtools.experiment.`**Experiment**(*z*, *line*, *energy_eV*, *kratio=0.0*, *standard=''*, *analyzed=True*)
Object to store experimental parameters and measurements. Once created an experiment object is immutable.

Parameters

- **z** (`int`) – atomic number
- **line** (`int`) – X-ray characteristic line, either
  - *LINE_KA*
  - *LINE_KB*
  - *LINE_LA*
  - *LINE_LB*
  - *LINE_MA*
  - *LINE_MB*

  Note that no other X-ray lines are supported.
- **energy_eV** (`float`) – beam energy (in eV)
- **kratio** (`float`) – measured k-ratio (optional)
- **standard** – three options
  - empty string for pure standard (e.g. `Fe`)
  - standard name which correspond to the filename of the standard saved in the standard directory (see `Stratagem.standard_directory`)
  - a `Sample`
- **analyzed** (`bool`) – whether to use this experiment in the calculations

is_analyzed()
Whether to use this experiment in the calculations.

z
Returns the atomic number.

line
Returns the x-ray characteristic line.

energy_eV
Return the beam energy (in eV).

kratio
Returns the measured k-ratio or `0.0` if no k-ratio was measured.

standard
REturns the standard.

2.3 Sample

Definition of a sample, corresponding to a multilayer. A multilayer a series of horizontal layers deposited on a substrate. A multilayer may also have no layer and simply be considered as a substrate. The substrate composition, and the composition, thickness and mass thickness of each layer are defined by the *Sample* class.

stratagemtools.sample.**CONC_UNKNOWN = None**
Flag when the composition of an element is unknown.

stratagemtools.sample.**CONC_DIFF = '?'**
Flag when the composition of an element should be calculated by difference.

stratagemtools.sample.**composition_from_formula**(*formula*)
Calculates the composition (expressed in weight fractions) of a chemical formula.

Example:

```
>>> composition_from_formula('Al2O3')
... {8: 0.4707492883573059, 13: 0.5292507116426941}
```

Parameters **formula** (str) – a valid chemical formula

Returns composition (expressed in weight fractions). The keys of the dict are atomic numbers and the values, weight fractions.

Return type dict

class stratagemtools.sample.**Layer**(*composition*, *thickness_m=None*, *mass_thickness_kg_m2=None*, *density_kg_m3=None*)
Object to store a layer definition. Once created a layer object is immutable.

Note: Should not be used directly, but via the *Sample*'s methods *add_layer* and *get_layer*.

is_composition_known()
Returns whether the composition is known, i.e. contains no *CONC_UNKNOWN* or *CONC_DIFF* flag.

Return type bool

is_thickness_known()
Returns whether the thickness is known.

Return type bool

is_density_known()
Returns whether the density is known.

Return type bool

composition
Returns a copy of the composition `dict`. The composition, and any other parameters, cannot be modified.

Returns composition (expressed in weight fractions). The keys of the `dict` are atomic numbers and the values, weight fractions.

Return type `dict`

thickness_m
Returns thickness in meters. It can be `None` if not defined.

Return type `float`

mass_thickness_kg_m2
Returns mass thickness in kg/m2. It can be `None` if not defined.

Return type `float`

density_kg_m3
Returns density in kg/m3. It can be `None` if not defined.

Return type `float`

class `stratagemtools.sample.`**Sample**(*composition*, *density_kg_m3=None*)
Object to store a multilayer sample definition.

Parameters

- **composition** (`dict`) – composition of the substrate as `dict` where the keys are atomic numbers and the values, weight fractions. If the weight fraction is not known, set it to *CONC_UNKNOWN*, if the weight fraction should be calculated by difference, set it to *CONC_DIFF*.
- **density_kg_m3** (`float`) – mass density in kilograms per cubic meter (optional). If the composition is known, the density will be automatically calculated based on:

$$\frac{1}{\rho} = \sum \frac{w_i}{\rho_i}$$

where w_i and ρ_i are respectively the weight fraction and mass density of element i.

add_layer(*composition*, *thickness_m=None*, *mass_thickness_kg_m2=None*, *density_kg_m3=None*)
Adds a layer below the previously added layer, or if no layer was added on top of the substrate.

Parameters

- **composition** (`dict`) – composition of the layer as `dict` where the keys are atomic numbers and the values, weight fractions. If the weight fraction is not known, set it to *CONC_UNKNOWN*, if the weight fraction should be calculated by difference, set it to *CONC_DIFF*.
- **thickness_m** (`float`) – thickness of the layer in meters (optional). If the *mass_thickness_kg_m2* and *density_kg_m3* are known, the thickness will be automatically calculated.
- **mass_thickness_kg_m2** (`float`) – mass thickness of the layer in kilograms per square meter (optional). The mass thickness is defined as the thickness times the density. If *thickness_m* and *density_kg_m3* are known the mass thickness will be automatically calculated.
- **density_kg_m3** (`float`) – mass density in kilograms per cubic meter (optional). If the composition is known, the density will be automatically calculated based on:

$$\frac{1}{\rho} = \sum \frac{w_i}{\rho_i}$$

where w_i and ρ_i are respectively the weight fraction and mass density of element i.

Returns a layer

Return type *Layer*

pop_layer(*index*)
Removes the layer at *index*.

Parameters **index** (int) – index of the layer to be removed

get_layer(*index*)
Returns the layer at *index*. Index *0* is the first layer of the multilayer, while index *-1* is the last layer, the first one above the substrate.

Parameters **index** (int) – index of the layer

Returns a layer

Return type *Layer*

composition
Returns the a copy of the composition of the substrate. The composition, and any other parameters, cannot be modified.

Returns composition (expressed in weight fractions). The keys of the dict are atomic numbers and the values, weight fractions.

Return type dict

substrate
Returns the "layer" of the substrate, a *Layer* object corresponding to the composition and density of the substrate.

Return type *Layer*

layers
Returns a copy of layers of this sample. It cannot be modified. The layers are ordered from top to bottom.

Return type tuple

2.4 Stratagem

Main class of the interface. It setups the experimental parameters such as the *Experiment*'s and *Sample*, geometry (*geometry*), type of $\phi(\rho z)$ model (*prz_mode*) and fluorescence mode (*fluorescence*).

stratagemtools.stratagem.**PRZMODE_XPP = 0**
$\phi(\rho z)$ from XPP

stratagemtools.stratagem.**PRZMODE_PAP = 1**
$\phi(\rho z)$ from PAP

stratagemtools.stratagem.**PRZMODE_GAU = 2**
$\phi(\rho z)$ *unknown*, possibly two Gaussians

stratagemtools.stratagem.**FLUORESCENCE_NONE = 0**
No fluorescence

stratagemtools.stratagem.**FLUORESCENCE_LINE = 1**
Only characteristic fluorescence

stratagemtools.stratagem.**FLUORESCENCE_LINE_CONT = 2**
Characteristic and Bremsstrahlung fluorescence

exception stratagemtools.stratagem.**StratagemError**
Exception raised for all errors related to the STRATAGem interface.

class stratagemtools.stratagem.**Stratagem**(*dll_path=None*, *display_error=True*)
Main interface establishing a connection to the STRATAGem OEM interface and perform calculations using SAMx's STRATAGem. It is highly recommended to use *Stratagem* as a context manager (i.e. `with` statement) to ensure that the connection to the DLL is properly closed. For instance:

```
>>> with Stratagem() as strata:
...     strata.prz_mode = PRZMODE_XPP
```

Otherwise the following series of method must be called:

```
>>> strata = Stratagem()
>>> strata.init()
>>> strata.prz_mode = PRZMODE_XPP
>>> strata.close()
```

Parameters

- **dll_path** (str) – complete path to the location of stratadllogger.dll (optional). If None, the path is found in the Windows registry under Software\SAMx\Stratagem\Configuration. If the DLL is not found a *StratagemError* is raised.
- **display_error** (bool) – whether to display a message dialog on error

init()
Initializes and setups STRATAGem. It does not have to be used if *Stratagem* is used as a context manager.

close()
Closes the connection to the STRATAGem DLL. It does not have to be used if *Stratagem* is used as a context manager.

reset()
Resets all parameters to the defaults, remove all experiments and sample.

set_sample(*sample*)
Sets the sample, which will be used in all subsequent calculations. Note that only one sample can be defined.

Parameters **sample** (Sample) – sample definition

get_sample()
Returns the current sample. It can correspond to the sample defined by *set_sample()* or the sample resulting from the computations (see *compute()*).

Note: a new sample is returned every time this method is called

---

Returns current sample

Return type Sample

sample
Property to set/get sample

add_experiment(*experiment*)
Adds an experiment, i.e. measurements of k-ratio at different energies.

Hint: Use *reset()* method to remove defined experiments.

Parameters **experiment** (Experiment) – experiment

add_experiments(**exps*)
Adds several experiments:

```
>>> strata.add_experiments(exp1, exp2, exp3)
```

get_experiments()
Returns a tuple of all defined experiments.

Return type tuple

set_geometry(*toa*, *tilt*, *azimuth*)
Sets the geometry.

Parameters

- **toa** – take off angle (in radians)
- **tilt** – tilt angle (in radians)
- **azimuth** – azimuthal angle (in radians)

get_geometry()
Returns the geometry.

Returns take off angle (in radians), tilt angle (in radians), azimuthal angle (in radians)

geometry
Property to get geometry

set_prz_mode(*mode*)
Sets the type of model to use for the $\phi(\rho z)$.

Parameters **mode** (int) – type of model, either

- *PRZMODE_XPP*
- *PRZMODE_PAP*
- *PRZMODE_GAU*

get_prz_mode()
Returns the type of model to use for the $\phi(\rho z)$.

Returns either *PRZMODE_XPP*, *PRZMODE_PAP* or *PRZMODE_GAU*

Return type int

prz_mode
Property to get/set prz mode

set_fluorescence(*flag*)
Sets the fluorescence flag.

Parameters **flag** (int) – either

- *FLUORESCENCE_NONE*

- *FLUORESCENCE_LINE*
- *FLUORESCENCE_LINE_CONT*

get_fluorescence()
Returns the fluorescence flag.

Returns either *FLUORESCENCE_NONE*, *FLUORESCENCE_LINE* or *FLUORESCENCE_LINE_CONT*

Return type int

fluorescence
Property to get/set fluorescence

set_standard_directory(*dirpath*)
Sets the directory where standard files are stored.

Parameters **dirpath** (str) – path to directory

get_standard_directory()
Returns the directory where standard files are stored.

Return type str

standard_directory
Property to get/set standard directory

compute_kratio_vs_thickness(*layer*, *thickness_low_m*, *thickness_high_m*, *step*)
Computes the variation of the k-ratio as a function of the thickness for a layer.

Parameters

- **layer** (*Layer*) – layer of a sample (must have been previously added)
- **thickness_low_m** (float) – lower limit of the thickness in meters
- **thickness_high_m** (float) – upper limit of the thickness in meters
- **step** (int) – number of steps

Returns

tuple containing

- list of thicknesses
- dict where the keys are experiments (as defined by *add_experiment()*) and the values are list containing k-ratios for each thickness

compute_kratio_vs_energy(*energy_high_eV*, *step*)
Computes the variation of the k-ratio as a function of the incident energy. Note that the computation also starts at 0 keV up to the specified energy.

Parameters

- **energy_high_eV** (float) – upper limit of the thickness in electronvolts
- **step** (int) – number of steps

Returns

tuple containing

- list of energies in electronvolts
- dict where the keys are experiments (as defined by *add_experiment()*) and the values are list containing k-ratios for each energy

compute_kratios()
Computes the k-ratios of the different experiments.

Returns dict where the keys are experiments (as defined by *add_experiment()*) and the values are k-ratios (float).

compute(*iteration_max=50*)
Computes the unknown composition(s) and thickness(es) in the specified sample.

Parameters **iteration_max** (int) – maximum number of iterations of the solve (default: 50)

Returns calculated sample

Return type *Sample*

compute_prz(*maxdepth_m=None*, *bins=100*)
Compute $\phi(\rho z)$ of all experiments.

Warning: Only available for substrate (no layers).

Parameters

- **maxdepth_m** (float) – maximum depth of the $\phi(\rho z)$ distribution in meters. If None, Kanaya-Okayama electron range is used with a safety factor of 1.5.
- **bins** (int) – number of bins in the $\phi(\rho z)$ distribution

Returns

a dict where the keys are the experiments and the values are a tuple containing three lists:

- ρz coordinates (in g/cm2)
- generated intensities of $\phi(\rho z)$ (no absorption)
- emitted intensites of $\phi(\rho z)$

CHAPTER 3

Tutorials

Here are some tutorials how to use *stratagemtools* for different applications.

3.1 Calculate k-ratios

This is a tutorial to calculate the Al, O and Si Kα k-ratios from a multilayer sample consisting of a Si substrate and 30-nm Al2O3 layer at an accelerating voltage of 15 kV.

Let's start by importing the three classes that will later need: *Sample*, *Experiment* and *Stratagem* classes.

```
from stratagemtools.sample import Sample
from stratagemtools.experiment import Experiment
from stratagemtools.stratagem import Stratagem
```

The *Sample* class is used to define the composition of the substrate and the composition, thickness and mass thickness of each layer. In this example, we first define the substrate composition as follows:

```
sample = Sample({14: 1.0})
```

The argument `{14: 1.0}` defines the substrate composition, consisting of silicon (atomic number 14) and with a weight fraction of 1.0 (pure silicon). To help us define the layer composition, we can use the utility function *composition_from_formula()* in the *sample* module. The function returns the composition expressed in weight fraction for a given chemical formula.

```
from stratagemtools.sample import composition_from_formula
comp = composition_from_formula('Al2O3')
```

The calculated composition can then be used to define a layer, using the method *add_layer*. The thickness and density (taken from Wikipedia) are also specified. Note that the thickness is expressed in meters and the density in kilograms per cubic meter.

```
sample.add_layer(comp, 30e-9, density_kg_m3=3950.0)
```

The next step is to define *Experiment*'s. *Experiment* specifies the experimental parameters used to analyze or to use to calculate the k-ratio of each element. As such, one experiment must be created for each element in the sample, whether or not it is analyzed or of interest to be calculated. For this example, the experiments are:

```
from stratagemtools.experiment import LINE_KA
energy_eV = 15e3
exp_si = Experiment(14, LINE_KA, energy_eV)
exp_al = Experiment(13, LINE_KA, energy_eV)
exp_o = Experiment(8, LINE_KA, energy_eV)
```

The constant *LINE_KA* is imported from the *experiment* module to specify the Kα X-ray line. For the moment, the standards (the denominator of the k-ratio) are all assumed to be pure sample, i.e. pure silicon, pure aluminum and (yes!) pure oxygen. The use of custom standards is addressed in another tutorial, *Custom standard*.

The `Sample` and `Experiment` objects should then be added to the *Stratagem* interface. The interface works as a context manager (`with` statement) in order to establish and properly close the connection to the STRATAGem's DLL. All operations on a sample and experiments should be performed inside the `with` statement. The following lines of code set the sample, add the experiment and compute the k-ratios.

```
with Stratagem() as strata:
    strata.set_sample(sample)
    strata.add_experiments(exp_si, exp_al, exp_o)
    kratios = strata.compute_kratios()
```

The *compute_kratios* method returns a `dict` where the keys are the experiments and the values, k-ratios. To help printing the results, the utility function `symbol` can be used to convert atomic number into element symbol.

```
import stratagemtools.element_properties as ep
for exp, kratio  in kratios.items():
    print('{0}: {1:.3f}'.format(ep.symbol(exp.z), kratio))
```

Going back to the `with` statement, it is advisable to always get the geometry, type of $\phi(\rho z)$ and fluorescence flag, as the default values may be changed. *stratagemtools* relies on the default values from STRATAGem.

```
import math
from stratagemtools.stratagem import PRZMODE_XPP, FLUORESCENCE_LINE_CONT
with Stratagem() as strata:
    strata.set_geometry(math.radians(40), 0.0, 0.0)
    strata.set_prz_mode(PRZMODE_XPP)
    strata.set_fluorescence(FLUORESCENCE_LINE_CONT)
```

3.2 Custom standard

This tutorial shows how to use different standards in the definition of the `Experiment`. The possibly to easily define standards is certainly an interesting feature of *stratagemtools*, in comparison to the graphical interface of STRATAGem where the standards must be manually defined and saved in separate files.

In *stratagemtools*, a standard is a *Sample*. Any sample definition can be used as a standard, as long as the composition and thickness of every layer are known. This example shows how to use three different standards to calculate the Al, O and Si Kα k-ratios from the previous tutorial, *Calculate k-ratios*.

We import the same packages and constants as the last tutorial

```
import math
from stratagemtools.sample import Sample, composition_from_formula
from stratagemtools.experiment import Experiment, LINE_KA
from stratagemtools.stratagem import Stratagem, PRZMODE_XPP, FLUORESCENCE_LINE_CONT
import stratagemtools.element_properties as ep
```

and create the unknown sample, a 30-nm Al2O3 layer over a Si substrate

```
    unknown = Sample({14: 1.0})
    comp = composition_from_formula('Al2O3')
    unknown.add_layer(comp, 30e-9, density_kg_m3=3950.0)
```

Now we define three standards, Fe2O3, SiO2 and Al2O3:

```
    std_fe2o3 = Sample(composition_from_formula('Fe2O3'), density_kg_m3=5240.0)
    std_sio2 = Sample(composition_from_formula('SiO2'), density_kg_m3=2650.0)
    std_al2o3 = Sample(composition_from_formula('Al2O3'), density_kg_m3=3950.0)
```

We then create experiments for Si and Al using the new standards. Note that in the previous tutorial pure standards were assumed.

```
    energy_eV = 15e3
    exp_si = Experiment(14, LINE_KA, energy_eV, standard=std_sio2)
    exp_al = Experiment(13, LINE_KA, energy_eV, standard=std_al2o3)
```

For the O Kα, all three standards can be used. To illustrate how easy it is to change the standard, we will loop over the standards and define a new experiment using each standard. This gives

```
    with Stratagem() as strata:
        strata.set_geometry(math.radians(40), 0.0, 0.0)
        strata.set_prz_mode(PRZMODE_XPP)
        strata.set_fluorescence(FLUORESCENCE_LINE_CONT)

        for std_name, std_o in [('Fe2O3', std_fe2o3),
                                ('SiO2', std_sio2),
                                ('Al2O3', std_al2o3)]:
            exp_o = Experiment(8, LINE_KA, energy_eV, standard=std_o)

            strata.reset()
            strata.set_sample(unknown)
            strata.add_experiments(exp_si, exp_al, exp_o)
            kratios = strata.compute_kratios()

            print(std_name)
            for exp, kratio  in kratios.items():
                print('{0}: {1:.3f}'.format(ep.symbol(exp.z), kratio))
            print('-' * 80)
```

3.3 Compute thickness

This tutorial shows how to use the main functionality of STRATAGem, to compute the mass thickness of a layer from experimentally measured k-ratios. For this example, we used the calculated k-ratios from the previous tutorial (*Custom standard*) as "experimental k-ratios". We should therefore compute a Al2O3 layer with a thickness of 30 nm.

As usual, let's start by importing the important classes and constants.

```
import math
from stratagemtools.sample import Sample, composition_from_formula
from stratagemtools.experiment import Experiment, LINE_KA
from stratagemtools.stratagem import Stratagem, PRZMODE_XPP, FLUORESCENCE_LINE_CONT
```

The next step is to define our *Sample*. Note here that we do not specify any thickness for the Al2O3 layer. No argument is given, which is equivalent to setting the thickness to None.

```
unknown = Sample({14: 1.0})
unknown.add_layer(composition_from_formula('Al2O3'), density_kg_m3=3950.0)
```

From the previous tutorial (*Custom standard*), the following Al and O Kα k-ratios were respectively calculated 0.034 and 0.058 using the Al2O3 standard. We now use these values to define the experiments. We do not need to know the k-ratio for the Si Kα, but we need to specify this element as not analyzed.

```
energy_eV = 15e3
exp_si = Experiment(14, LINE_KA, energy_eV, analyzed=False)
exp_al = Experiment(13, LINE_KA, energy_eV, kratio=0.034, standard=std_al2o3)
exp_o = Experiment(8, LINE_KA, energy_eV, kratio=0.058, standard=std_al2o3)
```

We then execute the method *compute* from the *Stratagem* interface to compute the unknown thickness. The method returns a new *Sample* object with the calculated thickness. Note that the new *Sample* object could also be retrieved from the method *get_sample* or the property *sample* after executing the *compute* method.

```
with Stratagem() as strata:
    strata.set_geometry(math.radians(40), 0.0, 0.0)
    strata.set_prz_mode(PRZMODE_XPP)
    strata.set_fluorescence(FLUORESCENCE_LINE_CONT)

    strata.set_sample(unknown)
    strata.add_experiments(exp_si, exp_al, exp_o)
    newsample = strata.compute()
```

Finally, we can print the calculated thickness.

```
thickness_m = newsample.get_layer(0).thickness_m
print('Thickness of first layer: {0:.2f} nm'.format(thickness_m * 1e9))
```

Getting back on our feet, we get 30.02 nm!

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

S

A

C

D

E

F

G

I

K

L

M

P

R

S

T

Z