doi:10.1017/S1431927613006107
Microsc. Microanal. 19 (Suppl 2), 2013

pyMonteCarlo: A Common Programming Interface for Running Identical Simulations using Different Monte Carlo Programs

Philippe T. Pinard[1], Hendrix Demers[2], Raynald Gauvin[2] and Silvia Richter[1]

[1] Central Facility for Electron Microscopy, RWTH Aachen University, Aachen, Germany.
[2] Department of Mining and Materials Engineering, McGill University, Montréal, Québec, Canada.

Several Monte Carlo programs have been developed to help microscopists predict and understand the results obtained from their measurements. Most of them are freely available on the web or with the permissions from the authors. The physical models and assumptions used in the calculation of the electron trajectories and x-ray generation differ from one code to another, as well as the possible sample geometries and results that can be extracted. To combine the advantages of different codes or to compare the effect of different physical models inevitably requires to manually create and run a new simulation for each Monte Carlo program and then to individually analyze each result.

In an effort to simplify the process of running Monte Carlo simulations with more than one program, a common programming interface was developed. It was written in Python using object-oriented programming. The source code and documentation are freely available under the GNU General Public License version 3 at http://pymontecarlo.bitbucket.org. Bindings with the following Monte Carlo programs are currently available: DTSA-II/NISTMonte Gemini [1], MONACO 3.0 [2], PENELOPE 2011 [3], Wincasino 2.48 [4] and WinXRay 1.4 [5]. The interface is, however, extendable to integrate other Monte Carlo programs, including personal or protected codes.

The interface was designed to have common input and output objects that are independent of any Monte Carlo code. Program-specific classes are responsible to create the file(s) required to run a simulation using a program and to import the results back into the common results format of the interface. The complete procedure is performed in six steps (Fig. 1). First, the simulation *options* should be defined. These include the beam parameters (energy, diameter, direction and initial position), the geometry (composition of all regions) and the physical models to be used if available. Simulation options also contain a series of detectors and limits. A detector is an abstract concept which defines the type of results expected from the simulation: x-ray intensity, x-ray spectrum, electron trajectories, backscattering coefficient, etc. The limits specify when the simulation should be stopped: after a certain time or number of electrons. A code excerpt of how to setup a simulation options is given in Fig. 2. The second step is the *converter* which ensures that the simulation options are valid for a specific Monte Carlo code. The *exporter* then saves them in the file format of this Monte Carlo code. After running the simulation, the *importer* has the reverse responsibility, to convert the result file(s) from this Monte Carlo code into the common results format used by the interface. The importer also conciliates the results to express them in the same units.

The simulation options and results are respectively stored using the extensible markup language (XML) and hierarchical data format (HDF5). Both are open file formats that can be read and modified in other programming languages, if needed. A command line interface to run several simulations using multiple processors is available and a graphical interface, similar to the one used in pyPENELOPE [6], is under development. pyMonteCarlo was successfully used to simulate all 826 experimental measurements of the Pouchou and Pichoir k-ratio database [7] with the aforementioned Monte Carlo programs.

References:
[1] N. W. M. Ritchie, Surf. Interface Anal **37** (2005), pp. 1006-1011.
[2] N. Ammann *et al*, Microbeam Analysis Conference Proceedings (1990), pp. 150-154.
[3] F. Salvat *et al*, "PENELOPE-2011: A code system for Monte Carlo simulation of electron and photon transport", OECD/NEA Data Bank, Issy-les-Moulineaux, France (2011).
[4] D. Drouin *et al*, Scanning **29** (2007), pp. 92-101.
[5] R. Gauvin *et al*, Microscopy & Microanalysis **12** (2006), pp. 49-64.
[6] P. T. Pinard *et al*, Microscopy and Microanalysis **16** (Suppl. 2) (2010), pp. 280-281.
[7] J. L. Pouchou and F. Pichoir in "Electron Probe Quantitation", ed. K. F. J. Heinrich and D. E. Newbury, Plenum Press, New York (1991), pp. 31-75.

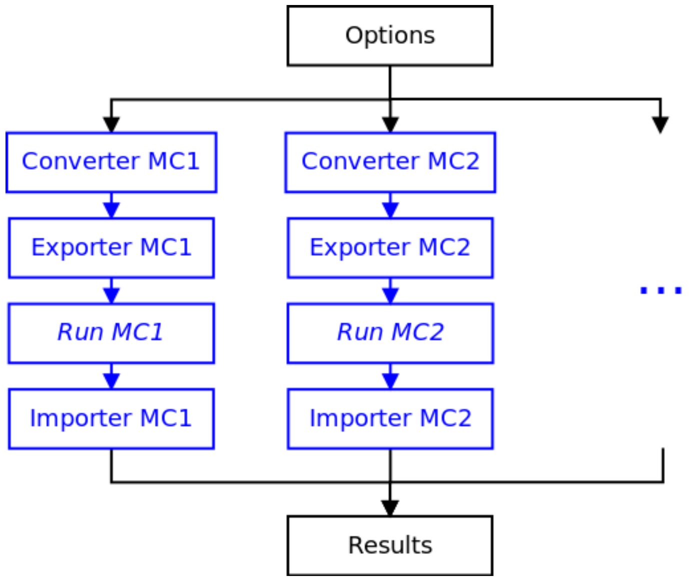

Figure 1: Schematic diagram of the interface showing how different Monte Carlo (MC) programs are integrated.

```
from pymontecarlo.input import *

ops = Options('simulation 1')
ops.beam.energy_eV = 5e3

ops.geometry = MultiLayers(Material('Brass', {30: 0.37, 29: '?'}))
ops.geometry.add_layer(pure(28), 500e-9) # 500 nm thick

elevation=(radians(35), radians(45)) # Take-off angle: 40 deg
azimuth=(0.0, radians(360)) # Annular detector
ops.detectors['intensity'] = PhotonIntensityDetector(elevation, azimuth)
ops.detectors['spectrum'] = \
    PhotonSpectrumDetector(elevation, azimuth, (0.0, ops.beam.energy_eV), 1000)

ops.limits.add(ShowersLimit(10000))
ops.models.add(ELASTIC_CROSS_SECTION.rutherford)

ops.save('simulation_1.xml')
```

Figure 2: Code excerpt showing how to setup the simulation options.