Acta Cryst A FOUNDATIONS ADVANCES

ISSN 2053-2733

Complex modeling: a strategy and software program for combining multiple information sources to solve ill posed structure and nanostructure inverse problems

Pavol Juhás,[a] Christopher L. Farrow,[b] Xiaohao Yang,[b] Kevin R. Knox[a] and Simon J. L. Billinge[a,b]*

[a]Condensed Matter Physics and Materials Science Department, Brookhaven National Laboratory, Upton, New York, 11973, USA, and [b]Department of Applied Physics and Applied Mathematics, Columbia University, New York, 10027, USA. *Correspondence e-mail: sb2896@columbia.edu

Received 17 May 2015
Accepted 31 July 2015

Edited by A. Altomare, Institute of Crystallography - CNR, Bari, Italy

Keywords: complex modeling; nanostructure analysis; Python software framework.

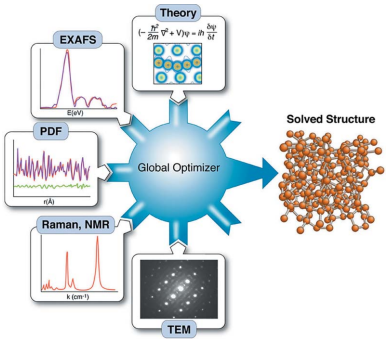
Supporting information: this article has supporting information at journals.iucr.org/a

A strategy is described for regularizing ill posed structure and nanostructure scattering inverse problems (*i.e.* structure solution) from complex material structures. This paper describes both the philosophy and strategy of the approach, and a software implementation, DiffPy Complex Modeling Infrastructure (*DiffPy-CMI*).

1. Introduction

In recent years, advances in materials synthesis techniques have enabled scientists to produce increasingly complex functional materials with enhanced or novel macroscopic properties. Modern engineered materials drive progress in many scientific fields and are at the heart of next-generation technologies in industrial fields including electronics (Dagotto *et al.*, 2003), energy production and storage (Rolison *et al.*, 2009), environmental engineering (Sun *et al.*, 2006), and biomedicine (Neuberger *et al.*, 2005). As the optical, electronic and mechanical properties of such materials are deeply influenced by atomic structure, solving the structure of engineered materials is of critical importance in unlocking their potential. However, the structure of such materials is often complex and non-periodic at the atomic scale or at the nanoscale. For example, many of the best known thermoelectric materials have structures that are crystallographic on average, but derive their high thermoelectric figure of merit from local atomic distortions (Mozharivskyj *et al.*, 2004; Hsu *et al.*, 2004; Biswas *et al.*, 2012; Lin *et al.*, 2005). Disordered collections of nanoparticles, on the other hand, have a high degree of short-range order, but no long-range order beyond the nanoscale (Steigerwald & Brus, 1989). Additionally, many novel materials are composites, which exhibit complex ordering on multiple length scales, such as core–shell nanoparticles or mesoporous materials, which are bulk materials with porous regions of nanoscale dimension that can be intercalated with a variety of structures (Stucky *et al.*, 2000; Armatas & Kanatzidis, 2006).

The standard techniques of crystallography have proven successful in characterizing a vast array of bulk materials whose atomic structures can be described with crystal models, which require only tens or hundreds of parameters. Since X-ray diffraction data typically yield information on hundreds or thousands of diffraction peaks a unique structure solution



can almost always be found for crystalline materials. However, for the type of complex materials described above the number of degrees of freedom in a suitable structure model is often considerably larger than in the case of a typical crystal. For example, surface atoms in nanoparticles often relax their atomic positions away from special crystallographic locations, thereby increasing the number of parameters needed to describe the overall nanoparticle structure (McGinley *et al.*, 2002; Zhang *et al.*, 2003). Additionally, complex engineered materials often produce extremely broad peaks in diffraction experiments, due to the fact that they are non-periodic or disordered. Thus, the structure problem is doubly complicated as diffraction experiments produce less information than corresponding experiments on bulk materials, despite the fact that more parameters are needed to describe their structures. From a standard crystallographic perspective, the structure problem for many complex materials is inherently ill posed, making a unique solution impossible (Billinge & Levin, 2007).

When the standard techniques of crystallography fail, it is sometimes possible to develop new analytical tools to maximize the information extracted from a diffraction pattern. One example is the pair distribution function (PDF) method, which uses both Bragg and diffuse scattering to simultaneously probe both local and long-range length scales. The PDF approach has proven successful in solving some nanostructure problems that are not solvable by direct inversion of single-crystal data; for example, the structure of C_{60} was solved using PDF analysis combined with *ab initio* algorithms (Juhás *et al.*, 2006; Cliffe *et al.*, 2010). However the PDF approach fails to obtain a unique solution of some disordered nanoparticles such as ultra-small whitelight CdSe (Yang *et al.*, 2013).

While such novel approaches can be helpful, a unique solution cannot be found to a fundamentally ill posed problem without defining new constraints or adding additional data. However, for many structure problems a single experimental technique cannot provide sufficient information to guarantee that the problem is well posed. To obtain unique structure solutions for complex materials, a new paradigm of analysis is needed, an infrastructure that can combine different information sources and models into a coherent framework to solve problems using global optimization. Within this framework, a material with unknown structure could be probed with various experimental tools, such as X-ray diffraction (XRD), transmission electron microscopy (TEM), small-angle X-ray or neutron scattering (SAS), Raman spectroscopy *etc.*, to yield an array of data sets that would then be fed into a global optimizer, as shown in Fig. 1. Additionally, theoretical inputs, such as density functional theory (DFT) or molecular dynamics calculations, as well as constraints on the variables coming from known symmetries or other sources, could be integrated into the optimization. While each single experimental or theoretical input may not generate enough information to produce a solution, together the pieces of information would regularize the problem resulting in a unique solution.

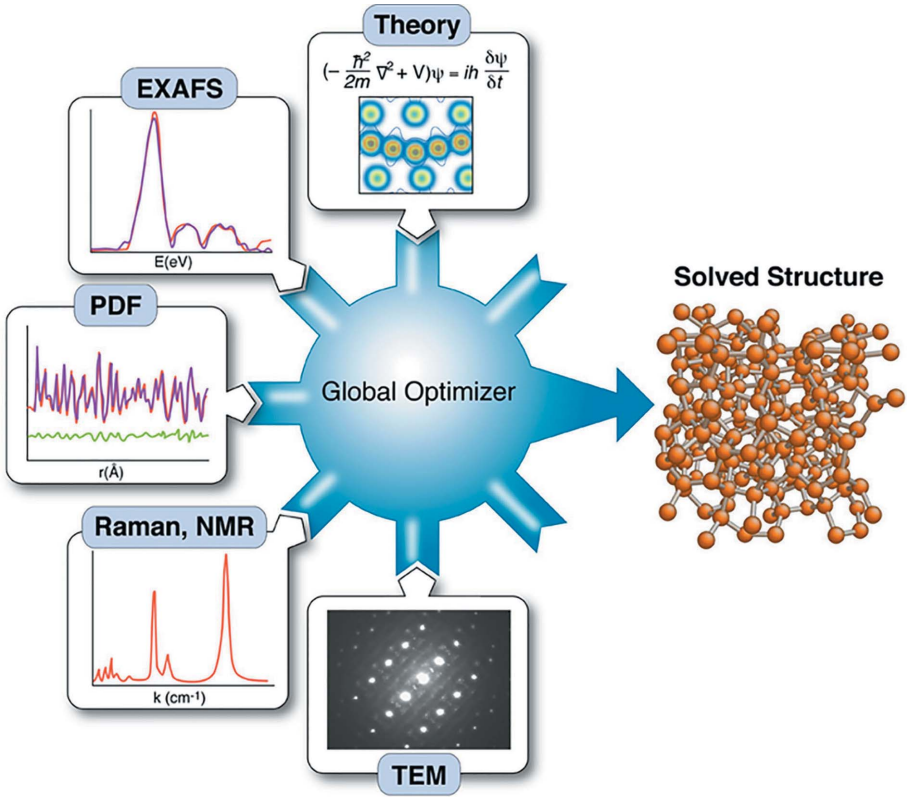


Figure 1
Complex modeling feeds all available data sets and theoretical constraints into a global optimizer to produce a unique structure solution for a new material.

In practice, structure solutions of complex materials are sometimes found by combining multiple techniques, but the work of integrating data streams is often arduous. Currently available software packages for data refinement are generally customized for one type of data. Thus, if a researcher has multiple sources of information, each data stream must be processed individually with a separate program and then manually combined with custom software to produce a co-refinement. However, the user group for such custom software is limited to individuals with both sophisticated programming skills and scientific knowledge across multiple fields. Additionally, many software suites for refinement of scientific data operate as 'black-boxes' which integrate the multiple steps required for solving a scientific problem into a single operation. While such integration may be useful for a novice researcher working in a single field, it further complicates the problem of co-refinement of multiple data sets.

In this paper we provide a complete description of an implementation of complex modeling, one which is robust,

modular, and easily adaptable to different types of problems and different combinations of data sets and theoretical inputs. The key is to break the process down into its constituent parts, which can then be combined and linked as necessary to solve the problem at hand.

2. Components of complex modeling

2.1. Structure model

The first step in solving the structure of any material is to choose a model representation, a way to describe the arrangement of atoms using a set of parameters and a mathematical formulation. In general, there is a wide range of methods to describe material structure and the most suitable choice will depend on the specifics of the system under investigation. In fact, the success of the refinement process itself can depend on how the structure is modeled; a good choice for the structure representation will simplify the process of model creation and help the refinement to converge quickly, while a poor choice can greatly complicate the modeling process or even cause the refinement to diverge. Thus, a proper implementation of a complex modeling framework must provide multiple options for representing atomic structures.

The simplest, though not necessarily the most useful, representation is a specification of atomic coordinates in a Cartesian coordinate system. A periodic crystal model, which extends the structure of one unit cell infinitely in all directions, is more suitable to describe many bulk materials. A nanocrystal structure model also extends one unit cell infinitely in all directions, but it applies an additional 'shape function' to the underlying structure to simulate the finite size of a nanocrystal (Proffen & Billinge, 1999; Farrow *et al.*, 2007; Farrow & Billinge, 2009); such a model is suitable for atomically well ordered nanoparticles. A molecular or cluster model specifies the position of each atom individually and is finite in extent; it is most suitable for describing molecules, disordered nanoparticles, and complex structures which cannot be described using a periodic model. Molecular systems may also be represented as a *Z*-matrix (Young, 2001). Additionally, as mentioned earlier, many novel materials exhibit complicated ordering on multiple length scales. To adequately describe such materials one may need to combine the simple building blocks described above to create hierarchical models of composite materials such as molecular crystals, nanosheets or mixtures of different phases.

Once the structure representation is specified, structure building tools are needed to build a particular instance of that structure. These should include basic operations such as importing and exporting structures from files, symmetry expansion and structure slicing, *i.e.* efficient selection and modification of its atoms.

Once a structure model is created, a suite of tools is required to modify and customize the model. Advanced operations, such as moving atoms under space-group constraints, should be possible. It should also be straightforward to impose restrictions on how atoms within the structure can move, such as restraints or constraints on bond lengths or bond angles.

The tools provided for operation on structure objects should derive naturally from the properties of those objects. For example, expanding the unit-cell volume of a simple crystalline structure by changing the lattice parameters should increase the distances between all atoms, but expanding the unit cell of a molecular crystal should affect only the separation between the molecules, not the distances between atoms in an individual molecule.

2.2. Function calculator

Ultimately, the connection between an atomic model and an experimental data set is made by calculating or simulating the data using the structure model. Thus, once a model is built, various quantities, which will later be fed into a refinement process, must be calculated by one or more calculator modules. Such a module takes the structure model as input and returns a physical quantity, which can be an experimentally measurable property, such as a PDF or XRD pattern, or some macroscopic property of the entire system, such as total energy or entropy.

One of the basic goals of complex modeling is to mix together different types of data and theoretical inputs to obtain a more complete description of a material system. Thus, an implementation of complex modeling must provide the capability to calculate multiple physical quantities and simulate various types of data. As each structure problem is unique and may have a different set of input data and theoretical constraints, the calculators themselves should be modular. Each structure problem can then be solved with a tailored approach by creating a custom 'complex', which links together the specific data sources and theoretical inputs available to the researcher.

2.3. Variables, constraints and restraints

A key aspect of complex modeling is the ability to seamlessly combine different structure models (in general with different representations) and different types of data into a single framework. This requires an effective strategy to manage different variables across multiple structure models and multiple data sets. Ideally, one set of parameters would be used to describe the atomic structure of each model, which would then be propagated through the function calculators to produce simulated data to be compared to the experimental data or theoretical results. Within the complex modeling framework, any parameter used in describing the structure of a material should be able to be passed as a refinable variable to the global optimizer. Once parameters are declared as variables it should be possible to turn them 'on' (allowed to vary) or 'off' (fixed) for a given refinement. Additionally, it should be straightforward to define constraints on variables in the form of mathematical relationships with other variables or model parameters. In general, for a particular set of variables A, B, C, . . ., the complex modeling infrastructure should

allow one to require $A = f(B, C, \ldots)$ where f is an arbitrary function.

Sometimes, additional information is available from a previous experiment or from a theoretical result; for example, that the value of one of the variables lies within a given range. To add this piece of information to the refinement it should be possible to apply restraints to certain variables. This is commonly done by adding a penalty to the refinement process commensurate with the deviation of the variable or variables from the known value or range.

2.4. Cost function

Once the models, calculators, variables and constraints have been defined, one must define a metric that quantifies the 'goodness-of-fit' of the structure model to the experimental data; such a metric is called a cost function. The choice of cost function will depend on the type and quality of data. Often the most appropriate function is the L^2-norm distance between the experimental and calculated profiles (Horn & Johnson, 1990). However, in general, the complex modeling framework should be flexible enough to allow for alternative metrics, such as an L^1-norm (Horn & Johnson, 1990). Additionally, if the refinement contains multiple models or multiple data sets, it should be possible to define several independent cost functions, which can then be weighted and combined to obtain a total cost function. Properly weighting the individual components in a multi-component refinement can be complicated and the ideal weighting may not be known *a priori*; thus a complex modeling framework must support dynamic weighting schemes.

2.5. Fit recipe

When properly defined and appropriately linked, the modules discussed above – a structure model, function calculator, set of constraints and a cost function – compose a recipe for fitting the model to the data. The fit recipe is simply a mathematical (or computational) object that takes a set of variables as input and returns a measure of goodness-of-fit as output. The process of fitting or refinement is then an optimization problem; one seeks the optimal set of input parameters to the fit recipe, which will produce the best goodness-of-fit.

2.6. Regression interface

The final step in constructing a complex modeling framework is choosing an algorithm to search the defined parameter space for a set of values which optimize the fit recipe. When called, the recipe will update the values of the variables as directed by the regression algorithm and return the goodness-of-fit. Normally, this process is repeated iteratively until some tolerance for the solution has been reached or a given number of iterations have been performed.

As discussed above, solving the structure of a complex material is often a highly non-trivial optimization problem; it is rarely possible to find a best fit to the data by a simple brute-force search of the variable space. Thus, choosing the correct algorithm to 'cook' the fit recipe is of particular importance; a simple least-squares algorithm may be appropriate for a fit with relatively few free variables and a good starting model, whereas a more complicated procedure, such as a Monte Carlo method, or an evolutionary algorithm, may be required for a recipe with a large number of variables. Viewed this way, the regression interface is simply another modular unit in the complex modeling framework that can be changed or adapted as necessary such that different regression algorithms may be inserted, or even nested together into a hybrid regression scheme.

3. Software implementation

The independent modules presented above define a complete picture of a complex modeling framework. In fact, any program or procedure for solving atomic structures must implement each. However, in practice, most software packages for data analysis bundle these modules together in a way that is not transparent and does not allow for user customization or extension; this is the 'black-box' approach discussed in §1. The promise of complex modeling can only be achieved when each part of the framework can be treated as an independent unit that can be modified by the user as the fit is set up; the ideal implementation of complex modeling would allow one to link these modules together in a customized way based on the problem at hand.

In this section we describe *DiffPy-CMI* (where CMI stands for Complex Modeling Infrastructure), our open-source Python-based object-oriented software solution for complex modeling. *DiffPy-CMI* follows the prescription laid out in the previous section for implementing a complex modeling framework: it is modular, flexible and extensible. It is composed of multiple Python modules, which can be linked together as necessary to solve structure problems. Computationally expensive functions are written in C++ with interface bindings to Python, allowing them to run at the speed of compiled code and allowing for a straightforward integration of useful components from other open-source projects such as *ObjCryst++* (Favre-Nicolin & Černý, 2002). Other open-source crystallography projects such as *CCTBX* (Grosse-Kunstleve *et al.*, 2002), *GSAS-II* (Dreele, 2013) and *CrysFML* (Rodríguez-Carvajal & González-Platas, 2003) also provide inspiration and methods for implementing functionality but, for various reasons, are not included as dependencies as part of *DiffPy-CMI*.

3.1. Extending the framework

Because of its inherent modular nature, *DiffPy-CMI* may be extended by writing new modules, such as new structure representations, function calculators and so on. Below we describe the existing capabilities at the time of writing, but the actual set of capabilities is rapidly changing with time. As we describe later, the software is open source and may also be maintained and extended by the community, with community-contributed modules and patches.

3.2. Structure representation

Structure modeling and manipulation are implemented within the *DiffPy-CMI* framework by the *diffpy.Structure* package, which provides objects for storage and handling of atomic coordinates, displacement parameters and other crystal structure data. *diffpy.Structure* supports the import and export of structure data using several common file formats such as Crystallographic Information File (CIF), Protein Data Bank (PDB) and xyz. It provides functions for conversion between fractional and absolute Cartesian coordinates, symmetry expansion from asymmetric units, and generation of symmetry constraints for atom positions and displacement parameters. Additionally, *diffpy.Structure* includes definitions of all space groups in over 500 symmetry settings, which were generated using the *sgtbx* module of *CCTBX* (Grosse-Kunstleve *et al.*, 2002).

Additional functionality for structure creation and manipulation is derived from *ObjCryst++*, an object-oriented crystallographic library developed by Vincent Favre-Nicolin (Favre-Nicolin & Černý, 2002). Certain portions of *ObjCryst++* have been repackaged and are distributed (with permission) along with *DiffPy-CMI*. To facilitate integration of the *ObjCryst++* library with the rest of *DiffPy-CMI* the *DiffPy* development team has written the *pyobjcryst* library, which provides Python bindings to much of the *ObjCryst++* functionality.

As described in the previous section, *DiffPy-CMI* allows the user to build and manipulate structures using the most appropriate models for the system under investigation, including periodic crystals, nanocrystals and molecules. Users can then build more complex structures by hierarchically combining these basic units. A simple demonstration of these features is available as an IPython notebook in the supporting information and also as a live notebook viewer (http://nbviewer.ipython.org/github/pavoljuhas/nb2015-ACA-CMI).

3.3. Function calculators

The *DiffPy-CMI* software package contains *diffpy.srreal*, which provides calculators for several pair-based quantities including PDF, bond-valence sums, atom overlaps for hard-sphere models, and bond distances and directions. The package provides implicit adapters from the *diffpy.Structure* class and from Crystal and Molecule objects from *pyobjcryst*. Additionally, adapters can be easily defined for any other structure representation in Python allowing their direct use with the calculators. Calculators support two evaluation models – Basic, which performs a full pair summation every time, and Optimized, which updates only pair contributions that have changed since the last evaluation. Calculations can be split among parallel jobs using the Python multi-processing package or any other library that provides a parallel map function. PDF calculations can be done in two modes – either as a real-space summation of peak profiles (using the PDFCalculator class) or as a reciprocal-space Debye summation with a Fourier transform of the total scattering structure function (using the DebyePDFCalculator class). The former is normally used for crystalline models with periodic boundary conditions or large box models with many atoms, whereas the latter is preferred for small, finite objects such as molecules or small nanoclusters.

The *diffpy.srreal* package is a Python binding to the C++ library *libdiffpy*. Calculators are created as objects of a given calculator type and so multiple instances of the same calculator type can exist with different configurations. Calculators are composed of other objects that perform lower-level tasks, such as calculating peak profiles or looking up atomic scattering factors. These objects can be re-assigned at runtime allowing the calculation procedure to be easily customized. New classes can be defined using object inheritance, either in Python or in C++, and used with the existing calculators; for example, a user can easily define a custom peak profile function to be used in PDF calculations. A new calculator class can also be defined for any quantity that is obtained by iteration over atom pairs, by defining only the function that processes atom-pair contributions, such as an interatomic pair potential. The capabilities of *diffpy.srreal* are highlighted in the supporting IPython notebook (http://nbviewer.ipython.org/github/pavoljuhas/nb2015-ACA-CMI).

3.4. *SrFit*

The remaining modules required for a complete complex modeling framework – variable control with constraints and restraints, the calculation of cost functions, the implementation of a fit recipe and of a regression interface – are found in the *SrFit* (*diffpy.srfit*) package. *SrFit* provides the framework for building a custom 'complex' by the user. It is designed to interface with *diffpy.Structure* for model representation and *diffpy.srreal* for calculation of PDFs and other pair-based quantities. However, in principle, any data stream can be used to define a profile to be fit and any custom function with well defined parameters and variables can be specified as the fitting function. *SrFit* makes it easy for users to define custom analytic functions to model their data. So, the framework defined by *SrFit* is not limited to atomic structure problems, but can be used to solve a wide variety of optimization problems (see IPython notebook, http://nbviewer.ipython.org/github/pavoljuhas/nb2015-ACA-CMI).

Once an *SrFit* recipe is defined it works just like a normal function and can be plugged into a number of regression algorithms. Currently, *SrFit* provides an interface that is compatible with *scipy.optimize* and *scipy.fmin* with extensions under development to work with other regression tools, such as genetic algorithms, simulated annealing and Bayesian methods. In this way, convergence may be checked by using multiple regression methods on the same model.

3.4.1. Co-refinement. As mentioned above, it is possible to integrate external calculators into *DiffPy-CMI* to perform co-refinements with other techniques. For example, *DiffPy-CMI* could be combined with a DFT calculator that accepts a structure model from *DiffPy-CMI*, calculates the energy of the structure and returns it to *DiffPy-CMI* to be included in the cost function calculation.

As discussed above, one drawback of currently available software packages for refinement is that most are limited to a specific field. *DiffPy-CMI* provides a platform to address this issue. *DiffPy-CMI* users are only required to write small amounts of code to adapt their existing software to work with *DiffPy-CMI* in order to run a co-refinement. The significantly reduced workload required to incorporate new functionality into *DiffPy-CMI* will make complex modeling more easily accessible to the scientific community.

4. *DiffPy* open-source software community

The final goal of *DiffPy-CMI* is to provide a flexible platform for heterogeneous solutions to inverse problems with particular emphasis on inverse structure problems. A key aspect of this goal is community involvement to direct the development of the software in a way that is responsive to the needs of the scientific users.

The development version of *DiffPy-CMI* has been widely tested and has already been used in a number of published scientific studies (Prill *et al.*, 2015; Beecher *et al.*, 2014; Ghidiu *et al.*, 2014; Shi *et al.*, 2014; Farrow *et al.*, 2013, 2014; Choi *et al.*, 2014; Doan-Nguyen *et al.*, 2014; Zhu *et al.*, 2012, 2014; Jacques *et al.*, 2013; ; Tyrsted *et al.*, 2012; Jensen *et al.*, 2012). A stable version of the *DiffPy-CMI* software suite was formally released in April 2014 and currently has an active and growing community of users and collaborators. Since release the software has been downloaded by over 300 users. Instructions for installing the software, as well as help getting started and links to the user community can be found at http://www.diffpy.org.

DiffPy-CMI is an open-source community programming project and all source code is available on GitHub at https://github.com/diffpy. We also provide hands-on tutorials and interactive examples for users to learn the software. API documentation is available for developers who are interested in contributing and adding additional functionality. The *DiffPy-CMI* repositories may be forked and pull requests sent to the development team to incorporate new code, or patches, into the kernel.

Another useful resource for the *DiffPy* user community is the CMI-Exchange, which is also hosted on GitHub at https://github.com/diffpy/cmi_exchange. The CMI-Exchange is a place for users to share *DiffPy-CMI* Python scripts – anything from simple functions and custom peak shapes to full fit recipes used in analyzing data. Useful scripts and functions can then be wrapped into IPython plug-ins and managed using IPython's extension management functions (IPython is the recommended Python shell for *DiffPy-CMI*).

5. Summary

Complex modeling is a general procedure for regularizing ill conditioned inverse problems that are inherent for many nano-sized and complex structures. It combines multiple information sources available from experiment and/or theory to provide more information to constrain solutions to the inverse problem. *DiffPy-CMI* is an open-source, community-developed software framework for conducting complex modeling. *DiffPy-CMI* provides a rich set of software objects for representing structure models, calculating physical quantities and assembling multi-component optimizations. *DiffPy-CMI* has been designed for maximum customization and for integration with other materials science codes to facilitate structure refinements that are tailored to the specifics of the studied materials. The *DiffPy-CMI* software is distributed at http://www.diffpy.org and the latest source codes are at https://github.com/diffpy.

Acknowledgements

Early development of the software was carried out under the DANSE software development project funded by the US National Science Foundation through award DMR-0520547. Since 2012, the project has been funded as Laboratory Directed Research and Development (LDRD) Program 12-007 (Complex Modeling) at Brookhaven National Laboratory, which is funded by the US Department of Energy Office of Basic Energy Sciences grant DE-SC00112704.

References

Armatas, G. S. & Kanatzidis, M. G. (2006). *Science*, **313**, 817–820.
Beecher, A. N., Yang, X., Palmer, J. H., LaGrassa, A. L., Juhas, P., Billinge, S. J. L. & Owen, J. S. (2014). *J. Am. Chem. Soc.* **136**, 10645–10653.
Billinge, S. J. L. & Levin, I. (2007). *Science*, **316**, 561–565.
Biswas, K., He, J., Blum, I. D., Wu, C., Hogan, T. P., Seidman, D. N., Dravid, V. P. & Kanatzidis, M. G. (2012). *Nature (London)*, **489**, 414–418.
Choi, J. J., Yang, X., Norman, Z. M., Billinge, S. J. L. & Owen, J. S. (2014). *Nano Lett.* **14**, 127–133.
Cliffe, M. J., Dove, M. T., Drabold, D. A. & Goodwin, A. L. (2010). *Phys. Rev. Lett.* **104**, 125501.
Dagotto, E., Burgy, J. & Moreo, A. (2003). *Solid State Commun.* **126**, 9–22.
Doan-Nguyen, V. V. T., Kimber, S. A. J., Pontoni, D., Reifsnyder Hickey, D., Diroll, B. T., Yang, X., Miglierini, M., Murray, C. B. & Billinge, S. J. L. (2014). *ACS Nano*, **8**, 6163–6170.
Dreele, R. V. (2013). *GSAS-II*. Crystallography data analysis software. https://subversion.xor.aps.anl.gov/trac/pyGSAS.
Farrow, C. L., Bediako, D. K., Surendranath, Y., Nocera, D. G. & Billinge, S. J. L. (2013). *J. Am. Chem. Soc.* **135**, 6403–6406.
Farrow, C. L. & Billinge, S. J. L. (2009). *Acta Cryst.* A**65**, 232–239.
Farrow, C. L., Juhás, P., Liu, J., Bryndin, D., Božin, E. S., Bloch, J., Proffen, T. & Billinge, S. J. L. (2007). *J. Phys. Condens. Matter*, **19**, 335219.
Farrow, C., Shi, C., Juhás, P., Peng, X. & Billinge, S. J. L. (2014). *J. Appl. Cryst.* **47**, 561–565.
Favre-Nicolin, V. & Černý, R. (2002). *J. Appl. Cryst.* **35**, 734–743.
Ghidiu, M., Naguib, M., Shi, C., Mashtalir, O., Pan, L., Zhang, B., Yang, J., Gogotsi, Y., Billinge, S. J. L. & Barsoum, M. W. (2014). *Chem. Commun.* **50**, 9517–9520.
Grosse-Kunstleve, R. W., Sauter, N. K., Moriarty, N. W. & Adams, P. D. (2002). *J. Appl. Cryst.* **35**, 126–136.
Horn, R. A. & Johnson, C. R. (1990). *Matrix Analysis.* Cambridge University Press.
Hsu, K. F., Loo, S., Guo, F., Chen, W., Dyck, J. S., Uher, C., Hogan, T., Polychroniadis, E. K. & Kanatzidis, M. G. (2004). *Science*, **303**, 818–821.
Jacques, S. D. M., Di Michiel, M., Kimber, S. A. J., Yang, X., Cernik, R. J., Beale, A. M. & Billinge, S. J. L. (2013). *Nat. Commun.* **4**, 2536.

Jensen, K. M. Ø., Christensen, M., Juhas, P., Tyrsted, C., Bøjesen, E. D., Lock, N., Billinge, S. J. L. & Iversen, B. B. (2012). *J. Am. Chem. Soc.* **134**, 6785–6792.
Juhás, P., Cherba, D. M., Duxbury, P. M., Punch, W. F. & Billinge, S. J. L. (2006). *Nature (London)*, **440**, 655–658.
Lin, H., Božin, E. S., Billinge, S. J. L., Quarez, E. & Kanatzidis, M. G. (2005). *Phys. Rev. B*, **72**, 174113.
McGinley, C., Riedler, M., Möller, T., Borchert, H., Haubold, S., Haase, M. & Weller, H. (2002). *Phys. Rev. B*, **65**, 245308.
Mozharivskyj, Y., Pecharsky, A. O., Bud'ko, S. & Miller, G. J. (2004). *Chem. Mater.* **16**, 1580–1589.
Neuberger, T., Schöpf, B., Hofmann, H., Hofmann, M. & von Rechenberg, B. (2005). *J. Magn. Magn. Mater.* **293**, 483–496.
Prill, D., Juhás, P., Schmidt, M. U. & Billinge, S. J. L. (2015). *J. Appl. Cryst.* **48**, 171–178.
Proffen, T. & Billinge, S. J. L. (1999). *J. Appl. Cryst.* **32**, 572–575.
Rodríguez-Carvajal, J. & González-Platas, J. (2003). *IUCr Comput. Commun. Newsl.* **1**, 50–58.
Rolison, D. R., Long, J. W., Lytle, J. C., Fischer, A. E., Rhodes, C. P., McEvoy, T. M., Bourg, M. E. & Lubers, A. M. (2009). *Chem. Soc. Rev.* **38**, 226–252.
Shi, C., Beidaghi, M., Naguib, M., Mashtalir, O., Gogotsi, Y. & Billinge, S. J. L. (2014). *Phys. Rev. Lett.* **112**, 125501.
Steigerwald, M. L. & Brus, L. E. (1989). *Annu. Rev. Mater. Sci.* **19**, 471–495.
Stucky, G., Sakamoto, Y., Kaneda, M., Terasaki, O., Zhao, D. Y., Kim, J. M., Shin, H. J. & Ryoo, R. (2000). *Nature (London)*, **408**, 449–453.
Sun, Y.-P., Li, X., Cao, J., Zhang, W. & Wang, H. P. (2006). *Adv. Colloid Interface Sci.* **120**, 47–56.
Tyrsted, C., Ørnsbjerg Jensen, K. M., Bøjesen, E. D., Lock, N., Christensen, M., Billinge, S. J. L. & Brummerstedt Iversen, B. (2012). *Angew. Chem. Int. Ed.* **51**, 9030–9033.
Yang, X., Masadeh, A. S., McBride, J. R., Božin, E. S., Rosenthal, S. J. & Billinge, S. J. L. (2013). *Phys. Chem. Chem. Phys.* **15**, 8480–8486.
Young, D. (2001). *Computational Chemistry: a Practical Guide for Applying Techniques to Real World Problems.* New York: Wiley-Interscience.
Zhang, H. Z., Gilbert, B., Huang, F. & Banfield, J. F. (2003). *Nature (London)*, **424**, 1025–1029.
Zhu, M., Farrow, C. L., Post, J. E., Livi, K. J. T., Billinge, S. J. L., Ginder-Vogel, M. & Sparks, D. L. (2012). *Geochim. Cosmochim. Acta*, **81**, 39–55.
Zhu, M., Northrup, P., Shi, C., Billinge, S. J. L., Sparks, D. L. & Waychunas, G. A. (2014). *Environ. Sci. Technol. Lett.* **1**, 97–101.