JOURNAL OF APPLIED CRYSTALLOGRAPHY

ISSN 1600-5767

Received 24 February 2023
Accepted 14 June 2023

Edited by J. Ilavsky, Argonne National Laboratory, USA

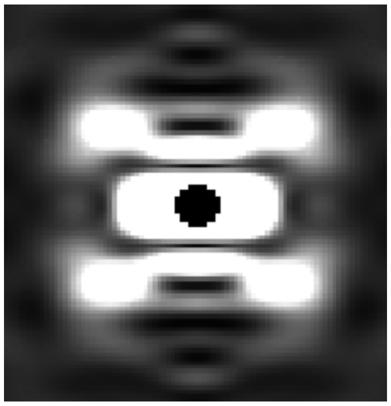
Keywords: X-ray scattering; fiber diffraction; hierarchical modeling; *D+* software; structure factors; polydispersity; orientations.

Upgrade of *D+* software for hierarchical modeling of X-ray scattering data from complex structures in solution, fibers and single orientations

Eytan Balken,[a] Itai Ben-Nun,[a] Amos Fellig,[a] Daniel Khaykelson[b] and Uri Raviv[a,c]*

[a]Institute of Chemistry, The Hebrew University of Jerusalem, Edmond J. Safra Campus, Givat Ram 9190401, Jerusalem, Israel, [b]Department of Molecular Chemistry and Materials Science, Weizmann Institute of Science, Rehovot 76100, Israel, and [c]Center for Nanoscience and Nanotechnology, The Hebrew University of Jerusalem, Edmond J. Safra Campus, Givat Ram 9190401, Jerusalem, Israel. *Correspondence e-mail: uri.raviv@mail.huji.ac.il

This article presents an upgrade of the *D+* software [Ginsburg *et al.* (2019). *J. Appl. Cryst.* **52**, 219–242], expanding its hierarchical solution X-ray scattering modeling capabilities for fiber diffraction and single crystallographic orientations. This upgrade was carried out using the reciprocal grid algorithm [Ginsburg *et al.* (2016). *J. Chem. Inf. Model.* **56**, 1518–1527], providing *D+* its computational strength. Furthermore, the extensive modifications made to the Python API of *D+* are described, broadening the X-ray analysis performed with *D+* to account for the effects of the instrument-resolution function and polydispersity. In addition, structure-factor and radial-distribution-function modules were added, taking into account the effects of thermal fluctuations and intermolecular interactions. Finally, numerical examples demonstrate the usage and potential of the added features.



1. Introduction

X-ray scattering is an important tool for determining molecular structures and intermolecular interactions. In an X-ray scattering experiment, the scattering intensity, I, is measured as a function of the scattering vector, $\mathbf{q}$, given by $\mathbf{q} = (q_x, q_y, q_z) = (q, \theta_q, \phi_q)$, in Cartesian and spherical coordinates, respectively. The scattering intensity is the square of the scattering amplitude, F, given by

$$F(\mathbf{q}) = -r_0 \int \Delta\rho(\mathbf{r}) \exp(i\mathbf{q}\cdot\mathbf{r})\,\mathrm{d}\mathbf{r}, \qquad (1)$$

where $\mathbf{r}$ is the position vector in real space; $\Delta\rho(\mathbf{r})$ is the electron-density contrast of the scattering particles, with respect to the medium, as a function of $\mathbf{r}$; and $r_0 = 2.82 \times 10^{-5}$ Å is the Thomson scattering length (Als-Nielsen & McMorrow, 2011). In solution, an orientation average over the reciprocal-space solid angle, Ω_q,

$$I(q) = \langle I(\mathbf{q})\rangle_{\Omega_q} = \frac{1}{4\pi}\int_{\Omega_q} \left|F(\mathbf{q})\right|^2 \mathrm{d}\Omega_q, \qquad (2)$$

should be computed. The term $\mathrm{d}\Omega_q = \sin\theta_q\,\mathrm{d}\theta_q\mathrm{d}\phi_q$, where θ_q and ϕ_q are the reciprocal ($\mathbf{q}$)-space polar and azimuthal angles, respectively.

Much effort has been devoted to scattering data analysis and modeling. With the evolution in the strength of computers came the evolution of X-ray technology, and the complexity of the experiments followed suit. As previously shown (Ginsburg *et al.*, 2019, 2016), the analysis program *D+* (https://scholars.huji.ac.il/uriraviv/book/d-0), developed in our laboratory, can accurately and rapidly compute the expected solution small/wide-angle X-ray scattering intensity from highly complex and large structural models.

In *D*+, structures can be defined in hierarchical data-structure trees, using geometric or atomic model subunits forming the tree's leaves. Repeating subunits are then docked into their assembly symmetries (the tree's nodes), containing the locations and orientations of repeating subunits. The scattering amplitude of the entire structure, made of J unique subunits, is

$$F(\mathbf{q}) = \sum_{j=1}^{J} \sum_{m=1}^{M_j^{\rm u}} \left[F_j\left(\mathbf{A}_{j,m}^{-1}\mathbf{q}\right) \sum_{k=1}^{K_{j,m}} \exp\left(i\mathbf{q}\cdot\mathbf{R}_{j,m,k}\right) \right]. \quad (3)$$

$M_j^{\rm u}$ is the number of unique orientations of an object of type j, given by the Tait–Bryan rotation matrices $\mathbf{A}_{j,m}$. Furthermore, $K_{j,m}$ is the number of real-space translations, $\mathbf{R}_{j,m,k}$, of object j with orientation $\mathbf{A}_{j,m}$. In *D*+, $F(\mathbf{q})$ can be calculated by computing the scattering amplitudes of the subunits on 3D reciprocal-space grids. When moving up in the hierarchy, the reciprocal grids of larger structures are computed by interpolating precomputed lower-level reciprocal grids. The final scattering amplitude is obtained by repeating this process for all the leaves and nodes of the tree data structure.

In the direct method, no grids are computed, and the scattering amplitude is directly computed by summing all the subunit contributions in the complex structure [using equation (3)]. For very large structures, a hybrid method can be used. In this method, only grids of smaller subunits are summed and used as subunits (*i.e.* leaves in the tree data structure) in a direct computation of the scattering amplitude [equation (3)] for the nodes in the hierarchy for which grids are not computed (Ginsburg *et al.*, 2019). The orientation average [equation (2)] is then numerically calculated by selecting many random angles in reciprocal space until the scattering intensity converges.

The *D*+ program has been used to analyze steady-state and time-resolved solution X-ray scattering data from various complicated structures at high resolution (Asor *et al.*, 2017, 2020*a*; Shaltiel *et al.*, 2019; Dharan *et al.*, 2021; Ginsburg *et al.*, 2017). In addition, to analyze an ensemble of dynamic structures and to take into account their stability, *D*+ has been integrated with Monte Carlo simulations (Louzon *et al.*, 2017; Asor *et al.*, 2019), the thermodynamic theory of macromolecular self-assembly (Asor *et al.*, 2019; Shemesh *et al.*, 2021), rate equations (Shemesh *et al.*, 2022) and maximum information entropy optimization (Asor *et al.*, 2020*b*).

Since the release of version 4.1, in 2019, several changes have been made to *D*+. We have upgraded the versions of CUDA (11.7) (https://www.nvidia.com/en-gb/geforce/technologies/cuda/), Python (3.8, 3.9, 3.10, 3.11), *Visual Studio* (2022) and *Ceres Solver* (2.0.0) (http://ceres-solver.org/), improved the error messages, created a more intuitive user interface, added GitHub automation to create the installer and the Python wheels, added tests, introduced JSON file format (https://www.json.org/), fixed several bugs, considerably improved the internal workflow of the program, and updated the Python API of *D*+ (https://scholars.huji.ac.il/uriraviv/book/python-api), making it independent of the installation of *D*+. We have also implemented the option to account for instrument resolution, similarly to how it is done in the *X*+ software (Ben-Nun *et al.*, 2010). Other changes, however, have taken *D*+ to the next level. We have built a Python API module that receives a list of N repeating subunit positions, $\mathbf{r}_i$ (*i.e.* a structural model), and computes the structure factor,

$$S\left(\mathbf{q}, N, \mathbf{r}_1, \ldots, \mathbf{r}_N\right) \equiv N^{-1} \sum_i^N \sum_j^N \exp\left[i\mathbf{q}\cdot\left(\mathbf{r}_i - \mathbf{r}_j\right)\right], \quad (4)$$

its orientation average,

$$S\left(q, N, \mathbf{r}_1, \ldots, \mathbf{r}_N\right) \equiv \left\langle S(\mathbf{q}, \mathbf{N})\right\rangle_{\Omega_q} = N^{-1} \sum_i^N \sum_j^N \frac{\sin\left(qr_{ij}\right)}{qr_{ij}}, \quad (5)$$

where $r_{ij} \equiv |\mathbf{r}_i - \mathbf{r}_j|$, and the radial distribution function,

$$g(r) \equiv \frac{N(r)}{4\pi\rho_{\rm b}r^2\Delta r}, \quad (6)$$

where $\rho_{\rm b}$ is the average bulk subunit number density. $N(r)$ is the number of repeating subunits in a shell of radius r and thickness Δr, from a random subunit r_i, averaged over all the subunits in the list, applying periodic boundary conditions. The module can also compute $g(r)$ from $S(q)$ and $S(q)$ from $g(r)$. Our most important upgrade is the option to compute the 2D scattering intensity pattern from a structure in a specific orientation, as typically measured by area detectors. In addition, we have created a module for computing the 2D fiber diffraction pattern from a fiber containing subunits with a uniform azimuthal angle distribution within a fiber aligned in a specific direction. This module performs orientation averaging over the reciprocal-space azimuthal angles, ϕ_q. Finally, in the Python API of *D*+, we have implemented the possibility of simulating the effects of thermal fluctuation and polydispersity in each parameter. All of these changes will be discussed and demonstrated in the following sections. The examples we shall present will provide insights into the functions and the correct usage of the Python API of *D*+.

2. Materials and methods

2.1. The Python API of *D*+

The Python API of *D*+ is a fully independent pythonic version of the *D*+ software, meaning that one can install the API without having to install the entire *D*+ software with its graphical user interface (GUI). Using the Python API of *D*+, the user can exploit and combine all the main functions of *D*+ as needed, and integrate them into any other code. The opposite is also true. One can integrate all the Python modules (like *NumPy* or *SciPy*; https://numpy.org/; https://scipy.org/) to build and simulate models, requiring advanced and sophisticated analyses.

The *D*+ program is broken down into different modules, providing all the computational functions of *D*+. Thus, we have the `CalculationInput` module, with which one can build the 'state file', containing all the information required for computing a model. In addition, the module `DataModels(.models)` covers all the structural models implemented in *D*+. After creating a model (and saving it in a state file), one calculates the scattering intensity of the model using the `CalculationRunner` module. In the end, if the model was run using a grid, one might want to use the `Amplitude` module for other purposes, like obtaining its 2D scattering pattern.

As the API has many functions, a document explaining all the inner workings of the Python API of *D*+ has been provided in our GitHub repository (API README; https://github.com/uri-raviv-lab/dplus-dev/blob/development/PythonInterface/README.md). Our code is open source for academic purposes, and fixes for encountered bugs or adding new features or functions are wholeheartedly accepted, as are any questions or problems encountered during installation or usage (*D*+ Issues, https://github.com/uri-raviv-lab/dplus-dev/issues; *D*+ GitHub, https://github.com/uri-raviv-lab/dplus-dev).

2.2. Resolution function

When performing an X-ray experiment, the scattering pattern might be different from what is computed from a model. One of the possible reasons is the finite resolution of the setup. We can take this effect into account, as done in the *X*+ program (Ben-Nun *et al.*, 2010), by convolving the modeled scattering intensity with a Gaussian resolution function, with a standard deviation given by σ. This function smears the model, meaning that some sharp peaks or minima will be less prominent or will merge with others (Pedersen *et al.*, 1990; Pauw, 2013). To simulate this effect, *D*+ has, next to the 'Generate' button, an input area for the standard deviation, σ, of a Gaussian instrument resolution function, offering users the choice of whether to take it into account or not. Similarly, a Gaussian resolution function with a specific σ can be added when building a state in the Python API of *D*+, using `apply_resolution`.

2.3. Polydispersity

In experiments, the measured particles often have a distribution of sizes rather than a single size. This effect can be modeled by adding a standard deviation to the geometric model parameters in the Python API of *D*+. In turn, as in *X*+ (Ben-Nun *et al.*, 2010), the model will be recalculated 14 times with a change of the size parameter, using the inputted standard deviation inside a Gaussian weighting distribution around the parameter's center value (*i.e.* 15 models will be calculated in total). This function only works with geometric models, not atomic ones. The polydispersity of atomic models might be computed by averaging the solution scattering intensity of different conformations generated, for example, by a Monte Carlo simulation (Louzon *et al.*, 2017).

Using the Python API of *D*+, any other polydispersity weighting function (Zimm, 1948; Kotlarchyk & Chen, 1983; Breßler *et al.*, 2015) can be implemented for geometric or atomic models.

2.4. $g(r)$ and $S(q)$ modules

The following subsections are based on the functions inside the $g(r)$ module, whose functions work, where relevant, with *NumPy* arrays (Harris *et al.*, 2020). The structure factor and the radial distribution function which are based on a structural model [equations (4)–(6)] have been parallelized on CPUs using the *DaCe* algorithm (Ben-Nun *et al.*, 2019). The parallel computation allowed us to get results in a reasonable time, even for large models.

2.4.1. Supporting functions.

In this module, three important supporting functions were built:

(1) `build_crystal` is a function similar to the space-filling symmetry, which builds a docking list ('dol') file from either lattice vectors $[\mathbf{a}, \mathbf{b}, \mathbf{c}]$ or lattice constants $[a, b, c, \alpha, \beta, \gamma]$, and the number of repetitions in each direction. By default, the crystal is moved to its geometric center.

(2) `thermalize` adds uniformly distributed random fluctuations to the lattice points according to $\Delta\mathbf{r} = 2\mathbf{u}v - \mathbf{u}$, where $v \in [0, 1]$ (randomly selected) and $\mathbf{u}$ is the maximal fluctuation in each coordinate. `thermalize` takes a list of points and adds normally distributed random fluctuations as defined by a user-inputted standard deviation. Unlike `build_crystal`, `thermalize` does not build a crystalline structure and then fluctuate the points, but rather takes an already built set of coordinates (that do not have to be crystalline) and adds fluctuations. In this way, it is possible to build different states that start from the same coordinates and differ only by a user-defined random factor.

(3) `MC_Sim` function is a Monte Carlo simulator, accounting for thermal fluctuations of a set of points (*i.e.* a dol file) by working against either harmonic or Lennard-Jones potentials (Jones, 1924*a*,*b*), $V(r)$, between nearest neighbors, nn. The pairwise energy cost, ΔE_i^j, of a random displacement $\mathbf{u}_i^j$ is

$$\Delta E_i^j = \frac{2}{\mathrm{nn_t}} \sum_{k\in \mathrm{nn}} V\left(|\mathbf{r}_i^{j-1} + \mathbf{u}_i^j - \mathbf{r}_k^{j-1}|\right) - V\left(|\mathbf{r}_i^{j-1} - \mathbf{r}_k^{j-1}|\right), \quad (7)$$

where $\mathrm{nn_t}$ is the total number of nearest neighbors. The factor of two originates from the fact that each interaction is shared between the interacting pair. The probability of obtaining a random displacement $\mathbf{u}_i^j$ is

$$P_i^j\left(\Delta E_i^j\right) \simeq \exp\left(-\frac{\Delta E_i^j}{k_{\mathrm{B}}T}\right), \quad (8)$$

where k_{B} is the Boltzmann constant and T is the absolute temperature. Monte Carlo simulations estimate the effect of thermal fluctuations. At iteration j, the probability, P_i^j, of a random displacement, $\mathbf{u}_i^j$, at a random lattice point $\mathbf{r}_i^{j-1}$ is compared against a random number between 0 and 1. The displacement is accepted if the random number is smaller than P_i^j. This process is repeated until the maximum number of iterations (provided by the user) is attained while maintaining periodic boundary conditions. At the end of the process, a new dol file is saved with the new coordinates.

2.4.2. Structure factor. The structure factor is an important property of crystals or oriented samples [$S(\mathbf{q})$] and solutions [$S(q)$]. It is needed for analyzing X-ray scattering data, as it sheds light on the arrangement of subunits in a sample (Yarnell *et al.*, 1973). Therefore, four functions were built to compute the structure-factor contribution, assuming monodispersed particles:

(i) `Amp_of_SF` receives a model as a dol file containing a list of subunit positions (a docking list generated, for example, using one of the supporting functions). The function computes the theoretical structure factor associated with the model in a single orientation [equation (4)].

(ii) `S_Q_from_model` receives a model as a dol file and computes the theoretical structure factor associated with the model after orientation averaging [equation (5)]. To `S_Q_from_model`, we added an option to include the effect of a finite temperature by averaging over N_T different random configurations (subunit positions, $\mathbf{r}_i$, using `thermalize`), assuming thermal fluctuations,

$$\Big\langle S\big(q, N, \mathbf{r}_1, \ldots, \mathbf{r}_N\big)\Big\rangle_{N_T} = \frac{1}{N_T}\sum_{j=1}^{N_T} S\big(q, N, \mathbf{r}_1 + \mathbf{u}_1^j, \ldots, \mathbf{r}_N + \mathbf{u}_N^j\big), \qquad (9)$$

where $\mathbf{u}_i^j$ is the jth displacement in the position $\mathbf{r}_i$ of the ith subunit. This function can better model realistic samples with thermal fluctuations. The displacements assume a Gaussian distribution of average standard deviation $\sigma_u = (\langle \mathbf{u}_i^j \cdot \mathbf{u}_i^j \rangle / 3)^{1/2}$.

(iii) `S_Q_from_I` computes the structure factor from the number of subunits, N, the total intensity, I, and the form factor of a subunit. The structure factor of oriented samples is a double sum over the complex exponents of the projections of the distances between all subunit pairs on the scattering vector (*i.e.* the phase factors) without the form-factor coefficients. The scattered intensity from an oriented sample containing copies of the same subunit is

$$\begin{aligned} I\big[\mathbf{q}, N, f(\mathbf{q}), \mathbf{r}_1, \ldots, \mathbf{r}_N\big] &= \sum_i^N \sum_j^N f_i(\mathbf{q}) f_j(\mathbf{q}) \exp(i\mathbf{q}\cdot\mathbf{r}_{ij}) \\ &= N f^2(\mathbf{q}) + f^2(\mathbf{q}) \sum_i^N \sum_j^N \exp(i\mathbf{q}\cdot\mathbf{r}_{ij}), \end{aligned} \qquad (10)$$

where the ith subunit form factor $f_i(\mathbf{q})$ is equal to $f(\mathbf{q})$ and $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, and the structure factor is given by equation (4). In isotropic solutions, we average over all the orientations and get a 1D structure factor [equation (5)] or

$$S\big(q, N, \mathbf{r}_1, \ldots, \mathbf{r}_N\big) = N + 2\sum_i^N \sum_{j>i}^N \frac{\sin\big(qr_{ij}\big)}{qr_{ij}} \qquad (11)$$

that is implemented in *D*+ because it is somewhat faster to compute.

Using the second function in equation (10), we can approximate the structure factor from a solution scattering curve, I, of a system with N identical subunits, provided we have the solution scattering curve of the subunit (form factor) $|f^2(q)|$:

$$S\big[q, N, I(q), |f^2(q)|\big] = \frac{I(q)}{N|f^2(q)|}. \qquad (12)$$

`S_Q_from_I` receives the number of subunits, N, the intensity and the subunit form factor (rather than a list of subunit positions), and computes equation (12). Therefore, the intensity must be correctly normalized to the absolute intensity and the density of subunits. Equation (12) assumes that the subunits are spherically symmetric. If not, deviations from the exact structure factor [equation (11)] appear (Fig. 1).

(iv) `s_q_from_g_r` converts a given radial distribution function, $g(r)$, into a structure factor using the known relation between the two (Als-Nielsen & McMorrow, 2011; Hansen & McDonald, 2013; Egelstaff, 1992):

$$S(q) = 1 + \frac{4\pi\rho_{\rm b}}{q}\int_0^\infty rg(r)\sin(qr)\,{\rm d}r. \qquad (13)$$

This function can be implemented using either *SciPy*'s discrete sine transform (DST) (Virtanen *et al.*, 2020) or the numerical Simpson integration method. Equation (13) has integration to infinity but the cut-off is determined by the $r_{\max}$ value of the radial distribution function.

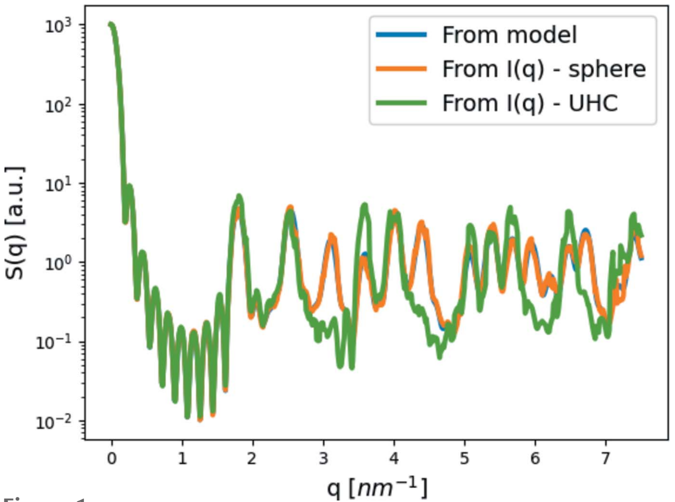


Figure 1
Examining the assumption behind structure factor, $S(q)$, computations. $S(q)$ of our cubic crystal model (lattice parameter a = 3.5 nm, and ten subunit repetitions in each axis) was computed in three ways. We first computed $S(q)$ from a list of subunit point locations [equation (11), blue curve]. The other two structure-factor curves were obtained using equation (12). Initially, the scattered intensity of the same crystal was computed, where the subunits were either symmetric (spheres with a radius of 1.5 nm) or asymmetric (cylinders with a height of 3.2 nm and a radius of 0.17 nm). We then divided each scattering intensity by its subunit form factor: sphere (orange curve) or cylinder (green curve). Except for some discrepancies owing to numerical errors, the difference between the crystal model (blue curve) and the symmetric subunit (sphere – orange curve) graph is minimal, unlike the difference between the crystal model (blue) and the asymmetric subunit (cylinder – green) graphs. The results demonstrate the assumption behind equation (12) stating that the form factor has to come from subunits with spherical symmetry.

In addition, using the Python API of *D*+, it is possible to compute any other structure-factor function (Baxter, 1970), fill an amplitude grid (using `fill`), and multiply it by the amplitude grid of the form factor (`Amp_multi`). The resulting amplitude can then be loaded to *D*+ or a state file and 'Generate' can be used to square the amplitude and compute its orientation average.

For polydispersed subunits, it is possible to implement equation (3) using the GUI of *D*+ or the Python API of *D*+. Equation (4) can be computed at each reciprocal grid amplitude point using the Python API of *D*+. The structure-factor amplitude grid can then be multiplied (`Amp_multi`) by any form-factor amplitude grid and added (`Amp_sum`) to other amplitude grids, representing other structures. The last two structure-factor options are demonstrated in a Jupyter Notebook (https://github.com/uri-raviv-lab/dplus-dev/blob/development/PythonInterface/getting_started.ipynb).

2.4.3. *g*(*r*). The radial distribution function is another tool often used to analyze X-ray data to study how local densities vary in a crystal or a liquid. From these data, one can understand the general structure and symmetries in a sample (Olgenblum *et al.*, 2020; Mu *et al.*, 2019; Yarnell *et al.*, 1973). Our module contains two functions:

(*a*) `g_r_from_s_q` computes the radial distribution function from a structure factor by reordering equation (13) and applying the inverse Fourier transform (Als-Nielsen & McMorrow, 2011; Biehl, 2019; Hansen & McDonald, 2013):

$$g(r) = \frac{1}{2\pi r \rho_{\rm b}} \int\limits_0^\infty q[S(q)-1]\sin(qr)\,{\rm d}q. \qquad (14)$$

This function encounters the same problems as its twin [equation (13)] and is computed up to a cut-off determined by the value of $q_{\max}$.

(*b*) `g_r_from_model` receives a model (a list of subunit positions, *i.e.* a dol file) and finds the radial distribution function through a simple binning technique around a randomly chosen point of reference [equation (6)]. This function can either count the number of subunits between *r* and *r* + Δ*r* (assuming the subunits are points) or take into account the subunit size (or radius) and compute the exact total volume of subunits between *r* and *r* + Δ*r*. It is possible to average different configurations (using one of the supporting functions) and/or different initial subunits of reference within the same configuration.

2.5. 2D scattering intensities

The 2D scattering intensity from oriented structures or fibers can be computed using the Python API of *D*+. First, one must calculate the 3D reciprocal grid scattering amplitude (using the GUI of *D*+ or its Python API). One can then use the Python API to calculate 2D scattering patterns from fibers or oriented structures (as in crystallography experiments). This can be done by positioning the structure in a specific orientation with respect to the *y* axis (which is the beam direction).

The user must also define the 2D pattern density by providing the total number of calculated points along each detector axis (from the negative to the positive side). This number is used for both the $q_\perp$ and q_z axes. In other words, the size of the returned 2D matrix will be the number of calculated points squared. The calculation time can thus quickly become very large. Interpolations are used to compute the amplitudes at points in the 2D matrix between precomputed reciprocal grid points.

In the current version of *D*+, to calculate 2D intensities one needs to use a reciprocal grid from the leaves up to the root of the model tree data structure. This means that the advantages of the hybrid method cannot be used yet, and large grids and reasonable computation power might be needed for computing the 2D scattering pattern from large structural models. This limitation will be removed in a future version of *D*+.

2.5.1. Single orientation. A function (`get_crystal_intensity`) has been written to simulate the 2D scattering intensity from a structure in a single orientation (as in crystallography experiments), whose input is a reciprocal grid amplitude computed by *D*+. Amplitudes are kept as a grid with polar coordinates in reciprocal space, $F(q, \theta_q, \phi_q)$. In experiments with oriented samples, the 2D detector intersects with the sample's Ewald sphere (represented by the amplitudes) at a specific ϕ_q. The default ϕ_q value is 0 for positive $q_\perp$ values and π for negative $q_\perp$ values. Other ϕ_q values can be calculated if needed, in which case, for negative $q_\perp$ values, $\phi_q + \pi$ is used instead of ϕ_q. The function gets the number of calculated points and generates the 2D matrix in $(q_\perp, q_z)$ space, and each pixel is converted to polar coordinates:

$$q = \left(q_\perp^2 + q_z^2\right)^{1/2} \qquad (15)$$

and

$$\theta_q = \operatorname{arctan2}\left(q_\perp, q_z\right). \qquad (16)$$

The 2D scattering intensity, $|F(q, \theta_q, 0)|^2$ for positive $q_\perp$ and $|F(q, \theta_q, \pi)|^2$ for negative $q_\perp$, is then obtained by interpolations from the 3D reciprocal grid of the root.

As the 3D reciprocal grid amplitude contains all the ϕ_q values, it is possible, using the same amplitude, to sample other planes of the structural model around the *z* axis by inputting a different ϕ_q value, $\phi_{q_{\rm in}}$, for positive $q_\perp$. In such a case, amplitudes at negative $q_\perp$ values are calculated at $\phi_q = \phi_{q_{\rm in}} + \pi$.

2.5.2. Fiber diffraction. In fiber diffraction experiments, the scatterers are aligned in the polar angle in real space, $\theta_{\rm r}$, and isotropically distributed around the azimuthal angle in real space, $\phi_{\rm r}$. The following azimuthal average, therefore, gives the scattering intensity in reciprocal space:

$$I\left(q, \theta_q\right) = \frac{1}{2\pi} \int\limits_0^{2\pi} \left|F(\mathbf{q})\right|^2 {\rm d}\phi. \qquad (17)$$

To compute this average, the API function `get_fiber_intensity` uses *D*+'s Monte Carlo integration function in which equation (16) becomes

$$I(q,\theta_q) = \frac{1}{N}\sum_{i=1}^{N}\left|F(q,\theta_q,\phi_{q_i})\right|^2, \tag{18}$$

and ϕ_{q_i} is randomly and uniformly sampled in the distribution $[0, 2\pi)$ by the use of the relationship $\phi = 2\pi u$ and $u \in [0, 1)$. The number of sampled ϕ_q values grows until convergence is attained according to the Monte Carlo algorithm of *D*+ (Ginsburg *et al.*, 2019). We also added the option to change the sampling domain to a user-defined one, simulating a sample with only certain azimuthal orientations. The sampling equation then becomes $\phi_{q_i} = (\phi_{\max} - \phi_{\min})u + \phi_{\min}$.

3. Usage examples

Several usage example codes and graphs can be found in our Jupyter Notebook. Some of the examples are discussed in this section.

3.1. Supporting functions

With these functions, we first built a simple cubic crystal with a lattice parameter a = 3.5 nm and ten repetitions in each axis (*i.e.* a 10 × 10 × 10 lattice), which will be the model used in our examples.

We then added thermal fluctuations to the same cubic crystal. We added displacements that are either uniformly distributed in the domain [−0.3 nm, 0.3 nm) [where the half-opened domain stems from the way random numbers are generated, as explained by Press *et al.* (2007)] or according to a Gaussian distribution with a standard deviation, σ_u, of 0.3 nm. In both cases, fluctuations were applied to all three axes. Fig. 2 shows a partial side view of the resulting crystals (shown as a 5 × 5 × 5 cube for clarity).

3.2. Resolution function

To demonstrate how the resolution function, computed as explained (Ben-Nun *et al.*, 2010), changes the scattering curve, we computed the scattering curves from spheres with a radius of 1.5 nm, arranged in our cubic crystal model. We calculated the scattering curve in *D*+, with and without a resolution function. We used typical σ_r values of 0.01 and 0.02 nm^{-1} but also some extreme values of 0.05 and 0.1 nm^{-1} that better

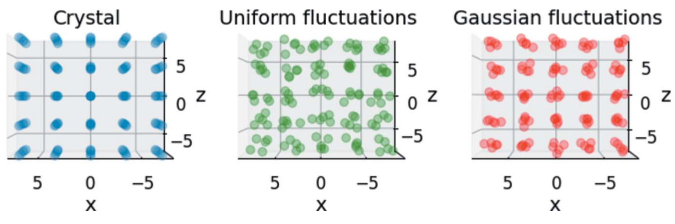


Figure 2
Our cubic crystal. (Left panel) A view of a 5 × 5 × 5 section of our simple cubic crystal model with a lattice parameter a = 3.5 nm and 10 × 10 × 10 subunits. This is the model used in our examples. (Center panel) A view of the same model, but, to each lattice point, a random fluctuation was added in all directions according to a uniform distribution within the domain [−0.3 nm, 0.3 nm). (Right panel) A view of a simple cubic crystal to which random fluctuations were added in all directions according to a Gaussian distribution with a mean at the lattice point coordinate and a standard deviation, σ_u, of 0.3 nm.

demonstrate the effect. As expected, the higher the σ_r, the less localized and prominent the peaks and minima are (Fig. 3).

3.3. Polydispersity

Similarly, we ran the same model (without a resolution function) and added a Gaussian polydispersity with σ_p of 0.3 nm around the mean 1.5 nm sphere radius. In other words, *D*+ computed multiple times the scattering intensity from spheres with radii selected from a Gaussian weighting distribution with a mean of 1.5 nm and a standard deviation of

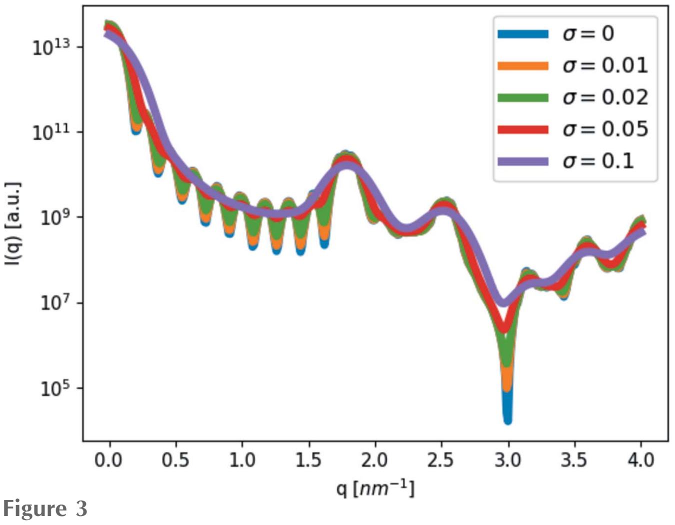


Figure 3
Effect of the instrument-resolution function. Scattering intensity, I, as a function of the magnitude of the scattering vector, q, from our simple cubic crystal (a = 3.5 nm and 10 × 10 × 10 subunits), with spheres of radius 1.5 nm as subunits, to which a Gaussian resolution function was added with four different standard deviation values: σ_r = 0.01, 0.02, 0.05 and 0.1 nm^{-1}. As σ_r grows, the peaks start to merge and the extrema become less localized.

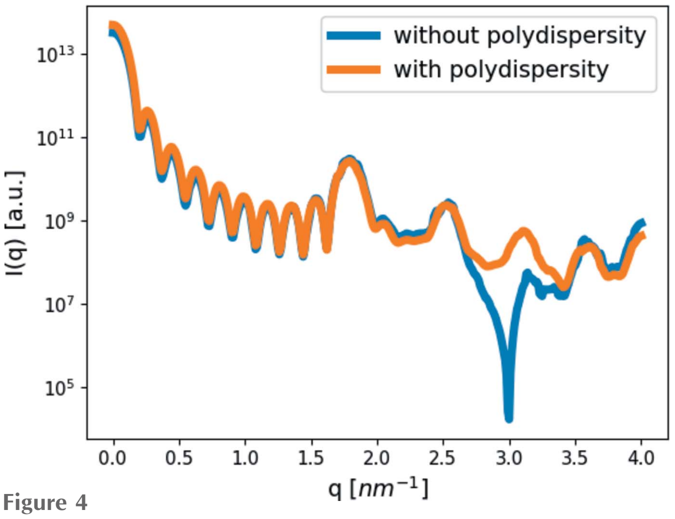


Figure 4
Effect of polydispersity. Scattering curves of our simple cubic crystal (a = 3.5 nm and 10 × 10 × 10 subunits), with spheres of radius 1.5 nm as subunits. We then added a Gaussian distribution function with a standard deviation of 0.3 nm to the radius of the spheres (with a mean at the inputted radius) to simulate the effect of polydispersity in the model. The minima, related to sphere radius, have shifted and are a lot less prominent, as would be expected from a polydispersed system.

0.3 nm (Fig. 4). In addition to filling the sharp minima as in Fig. 3, the positions of the minima and maxima have shifted slightly.

3.4. $S(q)$ and $g(r)$

This section demonstrates different ways to compute the structure factor and how they affect the result. In addition, we examined the effect the spherical symmetry assumption has on the result shown in Fig. 1. For this, we used a sphere with a radius of 1.5 nm as one model, and a cylinder with a height of 3.2 nm and a radius of 0.17 nm as a second model. We show that with the cylinders, already at $q \simeq 3\ \text{nm}^{-1}$, the assumption becomes critical for further analysis of the model.

Fig. 5 shows the calculated radial distribution function of our crystal [equation (6)], assuming the particles have a radius of 0.3 nm (using `g_r_from_model`), with and without thermal fluctuations (using `thermalize`). We compared it with the $g(r)$ calculated from our cubic crystal model, assuming the spheres are delta functions [Fig. 5(*b*)]. As expected, both radial distribution functions calculated from the model returned peaks at the typical distances for our cubic crystal, where the first three are at

$$\begin{cases} r_1 = a = 3.5\ \text{nm}, \\ r_2 = 2^{1/2}a = 4.95\ \text{nm}, \\ r_3 = 3^{1/2}a = 6.06\ \text{nm}. \end{cases} \tag{19}$$

However, when computing the radial distribution function from a structure factor [equation (14)], the results [Fig. 5(*c*)] are not as good as in the blue curve of Fig. 5(*b*). We get the peaks with Simpson's algorithm, but they are relatively wide and not always perfectly centered around the correct value. With the DST algorithm, we only get some of the peaks, whose maxima are sometimes off the expected values. It is, therefore, better to use Simpson's algorithm. Another difference between the model radial distribution function [equation (6)] and the integrated radial distribution function from a structure factor [equation (14)] is the intensity of the peaks. However, the area under the peaks,

$$N_{\rm NN} = \int_{r_1}^{r_2} 4\pi r^2 \rho_{\rm b}\, g(r)\, {\rm d}r, \tag{20}$$

revealing the total number of nearest neighbors (nn$_t$) (Als-Nielsen & McMorrow, 2011), is better preserved than the peak line-shape intensity. After integrating up from r_1 = 3.4 to r_2 = 3.6 nm, we found that the number of nearest neighbors (which is expected to be 6) is 5.90, 5.71 and 5.30 for the radial distribution function calculated from the model [equation (6)], Simpson's integration and the DST, respectively. Specifically for the DST, we integrated over the range [3.5, 3.7], because otherwise we got an incorrect $N_{\rm NN}$ owing to the negative oscillation before the peak.

Lastly, using equation (13), we tried to return to the structure factor [equation (5), Fig. 6, blue curve] from the radial distribution function calculated from a model [equation (6), Fig. 6, red curve] or a structure factor [equation (14), Fig. 6, green curve]. Fig. 6 shows that neither function was able to completely reproduce the modeled structure factor [equation (5), Fig. 6, blue curve]. However, when we took the structure

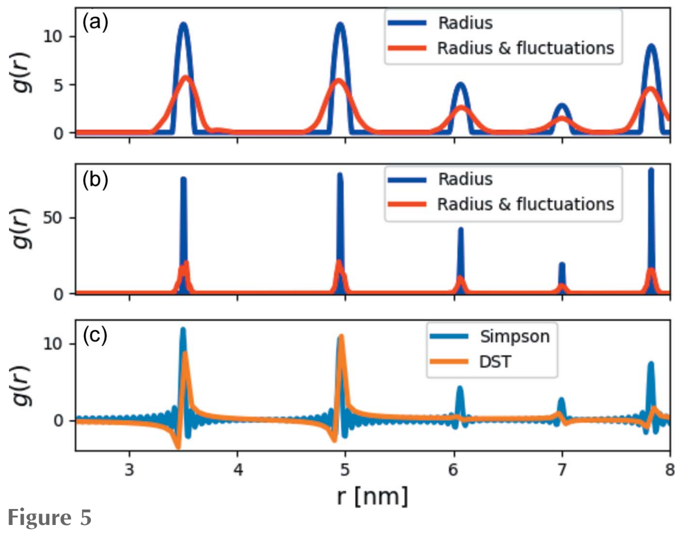


Figure 5
Radial distribution functions, $g(r)$. (*a*) The radial distribution function from a $10 \times 10 \times 10$ cubic crystal model (a = 3.5 nm), calculated according to equation (6) with spherical subunits with a radius of 0.1 nm, with (red curve) and without (blue curve) thermal fluctuations (σ_u = 0.1 nm). (*b*) Repeating the calculation in (*a*) with a subunit radius of 0.01 nm, with (red curve) and without (blue curve) thermal fluctuations (σ_u = 0.03 nm). We added a radius to the subunit lattice points to avoid delta functions at the peak positions and get results that are closer to reality. (*c*) The radial distribution function of the same crystal model (assuming the subunits are points), without thermal fluctuations, computed by equation (14) [using $S(q)$ from Fig. 6 (blue curve), computed until a $q_{\max}$ of 100 nm^{-1}], using Simpson's integration or DST.

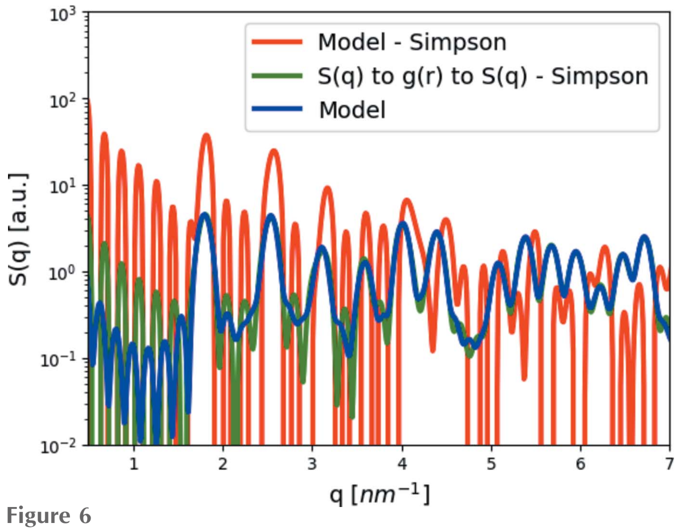


Figure 6
Comparing the structure factor emanating from the radial distribution function [equation (6), $r_{\max}$ = 32.5 nm] of a cubic crystal (a = 3.5 nm with $10 \times 10 \times 10$ subunits), using equation (13) and Simpson's integration (red curve), with the proper structure factor [equation (5), blue curve]. The green curve was obtained by taking the structure factor [equation (5), blue curve], Fourier transforming it into a radial distribution function [equation (14), using $q_{\max}$ = 8 nm^{-1}], and then changing it back to the structure factor, using Simpson's integration method [equation (13), using $r_{\max}$ = 32.5 nm].

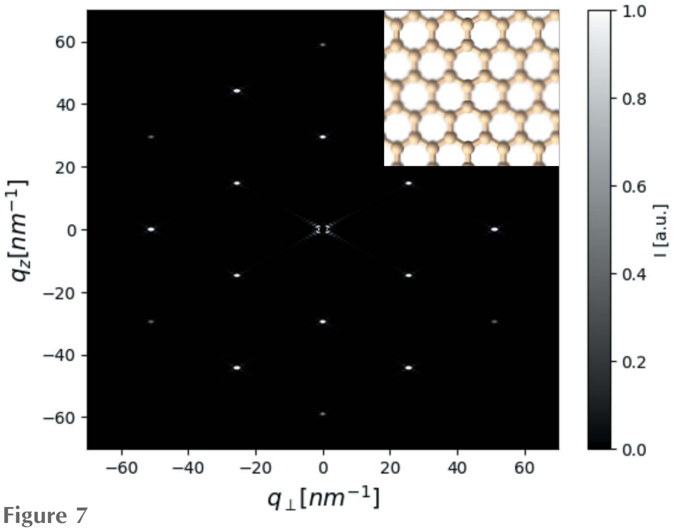


Figure 7
A simulated 2D scattering pattern from a layer of graphene (containing 25 × 25 unit cells; partly shown in the inset) aligned parallel to the *xz* plane (where the beam is along the *y* axis). The high intensity around $q = 0$ has been blacked out to emphasize the correlation peaks at higher *q* values. To make the different peaks more prominent, the intensities were rescaled using *scikit*'s (van der Walt *et al.*, 2014) `rescale_intensity` function. The simulated peaks fall on the peaks expected from the reciprocal lattice of graphene [equation (22)].

factor [equation (5)], Fourier transformed it to the radial distribution function [equation (14)] using Simpson's integration method, and then returned it to $S(q)$ [equation (13), Fig. 6, green curve] using the same integration method, the deviation from the modeled structure factor [equation (5), Fig. 6, blue curve] was rather small.

D+ allowed us to carefully examine the assumptions often applied when computing structure factors and radial distribution functions. We also demonstrated that care must be taken when analyzing the results as they can significantly deviate from the correct values.

3.5. Fiber diffraction and single orientation

To demonstrate the most important upgrade of the Python API of *D*+, we used a model of graphene and computed its 2D scattering pattern. This model was used because it is a 2D structure, and thus, the direction of the beam could be determined. Another reason was that the lattice parameters are known in real space and thus also in reciprocal space, meaning the resulting diffraction can be validated. The graphene model was built to be on the *xz* plane and used the following real-space lattice vectors (Wallace, 1947; Yang *et al.*, 2018):

$$\left\{\begin{array}{l}\mathbf{a}_1 = (a/2)\left[1, 0, 3^{1/2}\right],\\ \mathbf{a}_2 = (a/2)\,[0, 1, 0],\\ \mathbf{a}_3 = (a/2)\left[1, 0, -3^{1/2}\right],\end{array}\right. \tag{21}$$

where $a = 0.246$ nm. The crystal had 25 repetitions in the $\mathbf{a}_1$ and $\mathbf{a}_3$ directions and only one repetition (to get a 2D structure) in the $\mathbf{a}_2$ direction (which is parallel to the direction of the beam). The lattice vectors in reciprocal space are

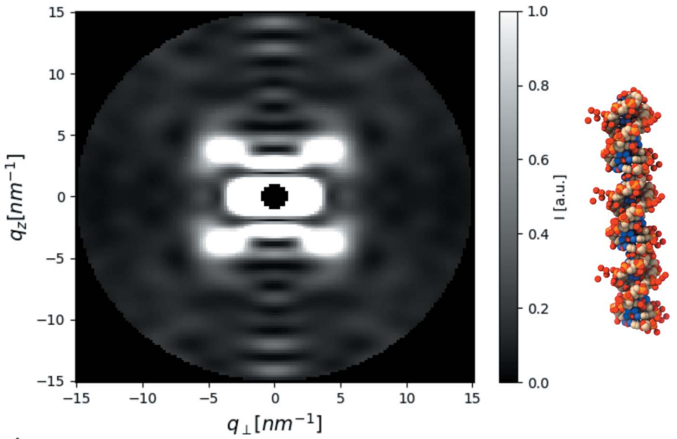


Figure 8
Computed fiber diffraction of a B-DNA model (using PDB ID 1zew). The typical 'X' 2D scattering pattern from double helices can be seen. The intensities were rescaled using the `rescale_intensity` function and a beamstop was added at the low *q* values.

$$\left\{\begin{array}{l}\mathbf{a}_1^* = (2\pi/a)\left[1, 0, 1/3^{1/2}\right],\\ \mathbf{a}_2^* = (2\pi/a)\,[0, 1, 0],\\ \mathbf{a}_3^* = (2\pi/a)\left[1, 0, -1/3^{1/2}\right].\end{array}\right. \tag{22}$$

Fig. 7 shows the 2D scattering pattern, which falls on the theoretically expected Bragg peaks. This demonstrates the single-orientation scattering simulation capabilities of *D*+.

To simulate a fiber diffraction experiment, we built a model whose base leaf is a PDB entry for DNA (Protein Data Bank; PDB ID 1zew; Hays *et al.*, 2005) inside a `Manual Symmetry` that added two other subunits, 3.5 nm above and 3.5 nm below the original unit. The PDB unit was aligned parallel to the *z* axis by rotating it with angles $\gamma = 60.5°$ around the *z* axis and $\alpha = 90°$ around the *x* axis. The result of the simulation (Fig. 8) clearly shows the expected typical 'X' pattern of helical structures (Cochran *et al.*, 1952). The computation time of a single orientation is about three orders of magnitude faster than the computation time of fiber diffraction (though this can vary with the convergence parameters of *D*+).

4. Conclusions

In this work, we have reported the functions and capabilities added to *D*+, some of which are implemented in the GUI and all in the Python API of *D*+. To the GUI (and the API), we added the possibility to add a resolution function to the resulting intensity and take into account the setup resolution. To the API, we added the possibility of taking into account the polydispersity of geometrical models. In addition, we added a module for computing the structure factor and the radial distribution function from either scattering data or a structural model, including the effects of thermal fluctuations and intermolecular interactions. Lastly, we added functions that can compute the 2D scattering pattern from either a single orientation or fibers. As *D*+ can create and compute highly complicated hierarchical structural models, these additions open new opportunities for modeling complex fiber and crystallographic structures. In future updates of *D*+, most changes will be implemented in the Python API of *D*+ owing to its wide versatility.

Acknowledgements

We are grateful for the helpful discussions with R. Beck, L. Houben, B. Rybtchinski, Y. Levi-Kalisman and D. Harries.

Funding information

E. Balken acknowledges the Heseg Foundation for supporting him throughout his undergraduate studies. This project was supported by the Israel Science Foundation (1331/20) and by the Israel Ministry of Science and Technology. We thank the Safra, Wolfson and Rudin Foundations for supporting our laboratory.

References

Als-Nielsen, J. & McMorrow, D. (2011). *Elements of Modern X-ray Physics.* John Wiley & Sons.
Asor, R., Ben-nun-Shaul, O., Oppenheim, A. & Raviv, U. (2017). *ACS Nano*, **11**, 9814–9824.
Asor, R., Khaykelson, D., Ben-nun-Shaul, O., Levi-Kalisman, Y., Oppenheim, A. & Raviv, U. (2020*a*). *Soft Matter*, **16**, 2803–2814.
Asor, R., Schlicksup, C. J., Zhao, Z., Zlotnick, A. & Raviv, U. (2020*b*). *J. Am. Chem. Soc.* **142**, 7868–7882.
Asor, R., Selzer, L., Schlicksup, C. J., Zhao, Z., Zlotnick, A. & Raviv, U. (2019). *ACS Nano*, **13**, 7610–7626.
Baxter, R. J. (1970). *J. Chem. Phys.* **52**, 4559–4562.
Ben-Nun, T., de Fine Licht, J., Ziogas, A. N., Schneider, T. & Hoefler, T. (2019). *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York: Association for Computing Machinery.
Ben-Nun, T., Ginsburg, A., Székely, P. & Raviv, U. (2010). *J. Appl. Cryst.* **43**, 1522–1531.
Biehl, R. (2019). *PLoS One*, **14**, e0218789.
Breßler, I., Kohlbrecher, J. & Thünemann, A. F. (2015). *J. Appl. Cryst.* **48**, 1587–1598.
Cochran, W., Crick, F. H. & Vand, V. (1952). *Acta Cryst.* **5**, 581–586.
Dharan, R., Shemesh, A., Millgram, A., Zalk, R., Frank, G. A., Levi-Kalisman, Y., Ringel, I. & Raviv, U. (2021). *ACS Nano*, **15**, 8836–8847.
Egelstaff, P. A. (1992). *An Introduction to the Liquid State*, p. 390. Oxford: Clarendon Press.
Ginsburg, A., Ben-Nun, T., Asor, R., Shemesh, A., Fink, L., Tekoah, R., Levartovsky, Y., Khaykelson, D., Dharan, R., Fellig, A. & Raviv, U. (2019). *J. Appl. Cryst.* **52**, 219–242.
Ginsburg, A., Ben-Nun, T., Asor, R., Shemesh, A., Ringel, I. & Raviv, U. (2016). *J. Chem. Inf. Model.* **56**, 1518–1527.
Ginsburg, A., Shemesh, A., Millgram, A., Dharan, R., Levi-Kalisman, Y., Ringel, I. & Raviv, U. (2017). *J. Phys. Chem. B*, **121**, 8427–8436.
Hansen, J.-P. & McDonald, I. R. (2013). *Theory of Simple Liquids: with Applications to Soft Matter*, 4th ed., pp. 1–619. Oxford: Academic Press.
Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E. (2020). *Nature*, **585**, 357–362.
Hays, F. A., Teegarden, A., Jones, Z. J., Harms, M., Raup, D., Watson, J., Cavaliere, E. & Ho, P. S. (2005). *Proc. Natl Acad. Sci. USA*, **102**, 7157–7162.
Jones, J. E. (1924*a*). *Proc. R. Soc. London Ser. A*, **106**, 441–462.
Jones, J. E. (1924*b*). *Proc. R. Soc. London. Ser. A*, **106**, 463–477.
Kotlarchyk, M. & Chen, S. (1983). *J. Chem. Phys.* **79**, 2461–2469.
Louzon, D., Ginsburg, A., Schwenger, W., Dvir, T., Dogic, Z. & Raviv, U. (2017). *Biophys. J.* **112**, 2184–2195.
Mu, X., Mazilkin, A., Sprau, C., Colsmann, A. & Kübel, C. (2019). *Microscopy*, **68**, 301–309.
Olgenblum, G. I., Sapir, L. & Harries, D. (2020). *J. Chem. Theory Comput.* **16**, 1249–1262.
Pauw, B. R. (2013). *J. Phys. Condens. Matter*, **25**, 383201.
Pedersen, J. S., Posselt, D. & Mortensen, K. (1990). *J. Appl. Cryst.* **23**, 321–333.
Press, W. H., Teukolsky, S. A., Vettering, W. T. & Flannery, B. P. (2007). *Numerical Recipes: the Art of Scientific Computing*, 3rd ed. Cambridge University Press.
Shaltiel, L., Shemesh, A., Raviv, U., Barenholz, Y. & Levi-Kalisman, Y. (2019). *J. Phys. Chem. C*, **123**, 28486–28493.
Shemesh, A., Ginsburg, A., Dharan, R., Levi-Kalisman, Y., Ringel, I. & Raviv, U. (2021). *ACS Chem. Biol.* **16**, 2212–2227.
Shemesh, A., Ginsburg, A., Dharan, R., Levi-Kalisman, Y., Ringel, I. & Raviv, U. (2022). *J. Phys. Chem. Lett.* **13**, 5246–5252.
Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., Vijaykumar, A., Bardelli, A. P., Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C. N., Fulton, C., Masson, C., Häggström, C., Fitzgerald, C., Nicholson, D. A., Hagen, D. R., Pasechnik, D. V., Olivetti, E., Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G. A., Ingold, G., Allen, G. E., Lee, G. R., Audren, H., Probst, I., Dietrich, J. P., Silterra, J., Webber, J. T., Slavič, J., Nothman, J., Buchner, J., Kulick, J., Schönberger, J. L., de Miranda Cardoso, J. V., Reimer, J., Harrington, J., Rodríguez, J. L. C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kümmerer, M., Bolingbroke, M., Tartre, M., Pak, M., Smith, N. J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb, P. A., Lee, P., McGibbon, R. T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson, S., More, S., Pudlik, T., Oshima, T., Pingel, T. J., Robitaille, T. P., Spura, T., Jones, T. R., Cera, T., Leslie, T., Zito, T., Krauss, T., Upadhyay, U., Halchenko, Y. O. & Vázquez-Baeza, Y. (2020). *Nat. Methods*, **17**, 261–272.
Wallace, P. R. (1947). *Phys. Rev.* **71**, 622–634.
Walt, S. van der, Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T. & The scikit-image Contributors (2014). *PeerJ*, **2**, e453.
Yang, G., Li, L., Lee, W. B. & Ng, M. C. (2018). *Sci. Technol. Adv. Mater.* **19**, 613–648.
Yarnell, J. L., Katz, M. J., Wenzel, R. G. & Koenig, S. H. (1973). *Phys. Rev. A*, **7**, 2130–2144.
Zimm, B. H. (1948). *J. Chem. Phys.* **16**, 1093–1099.