D+

User's Manual (Version 4.1.0)

Contents

Preface

D+ is a program that computes the X-ray scattering from large supramolecular structures in solutions at any resolution down to (near) atomic resolution. This document will focus on explaining how to use the software and very little of the inner workings (just enough to understand the options within the program). A paper that presents and discusses the principle algorithm of the program, the reciprocal-grid (RG) algorithm, is published elsewhere (Ginsburg et al. 2016). The implementation of the RG algorithm and several other algorithms and features in D+, as well as usage examples, extensive tests, and cross validations are published in Ginsburg et al. 2018. D+ is available for academic use.

Chapter 1

Installation

Computers themselves, and software yet to be developed, will revolutionize the way we learn.

Steve Jobs

1.1 Hardware architecture

D+ requires that the CPU is Intel Advanced Vector Extensions (Intel AVX) capable. D+ can also use graphics cards (or graphic processing units, GPUs) to accelerate calculations by factors that vary between $\times 10$ to $\times 1000$. A GPU is not required, but recommended for performance. D+ utilizes NVIDIA GPUs, preferably the higher end products with reasonable (1:3) double- to single- precision floating point ratio (FP64 = $1/3$ FP32) performance. The cards that are known/assumed to work well are

1. GTX 980 (known)
2. Titan (known)
3. Titan Black (known)
4. Titan Z (assumed)
5. Tesla series (assumed)
6. Quadro K6000 (assumed)
7. Quadro K5200 (assumed)

Cards that have low FP64:FP32 performance (1:32 for Maxwell architecture and therefore are not recommended, but will work for most computations) include:

1. Titan X
2. Quadro M6000

1.2 Software

Windows

D+ is freely available for academic use. You can download the installer of D+ and install the program.

D+ has a few prerequisites: Microsoft .NET 4.0 and Visual C++ Redistributable Packages for Visual Studio 2013. D+ is a 64 bit application due to memory usage and some library requirements; install the 64 bit versions of the Visual C++ Redistributable Packages. Also, if a GPU is installed, the proper driver should also be installed to enable CUDA (CUDA 8.0, at the time of writing). If the D+ installer (an msi file) is used, it will install the Visual C++ Redistributable Packages on its own. On Windows computers with CUDA enabled GPUs, the installer makes sure that the watchdog timer is not enabled.

Once all the above are installed, the files listed in Table 1.2 should be placed in the same directory (the default is `C:\Program Files\D+\bin`). The purposes for including some of these files are indicated Table 1.2. There are differences in the version that uses a remote server for the computations. Usage of both versions is practically identical and therefore except for the few and minor differences mentioned in chapter 9, the remainder of this manual pertains to both.

Linux

First note that there is no GUI for Linux. There are precompiled binaries for Ubuntu and CentOS that can be used by the Python API (see Chapter 2.14). Download the relevant file (`dplus.tar.bz2` or `dplus.centos.tar.bz2`) and `install.sh`. Run `$ install.sh` alongside the downloaded archive. Run `$ source ~/.bashrc` to set the proper environment variables.

Filename	**Purpose**
`Bin/Aga.Controls.dll`	
`Bin/ceres.dll`	
`Bin/cudart64_80.dll`	Checks for the graphics card
`Bin/curand64_80.dll`	
`Bin/DebyeCalculator.exe`	Calculates the "ground truth" for atomic scattering
`Bin/DPlus.exe`	Launches D+
`Bin/Fit.exe`	
`Bin/Generate.exe`	
`Bin/GetAllMetadata.exe`	
`Bin/GLView.dll`	
`Bin/GraphToolkit.dll`	
`Bin/JSON.lua`	
`Bin/License.txt`	
`Bin/lua51.dll`	
`Bin/lua51-backend.dll`	
`Bin/LuaInterface.dll`	
`Bin/Microsoft.WindowsAPICodePack.dll`	
`Bin/Microsoft.WindowsAPICodePack.Shell.dll`	
`Bin/mscorlib.dll`	
`Bin/Newtonsoft.Json.dll`	
`Bin/PDBReaderLib.dll`	
`Bin/PDBUnits.exe`	Helper tool that finds identical copies of segments in a larger PDB file
`Bin/Resources/atomicData.txt`	Used by `Bin/PDBUnits.exe`
`Bin/SciLexer.dll`	
`Bin/SciLexer64.dll`	
`Bin/ScintillaNET.dll`	
`Bin/Suggest Parameters.exe`	A program that suggests parameters for D+
`Bin/WeifenLuo.WinFormsUI.Docking.dll`	
`Bin/xplusbackend.dll`	Handles the actual calculations
`Bin/xplusfrontend.dll`	
`Bin/xplusmodels.xrn`	
`Example Files/Examples`	Contains 7 Example subfolders. See Chap. 8.
`Example Files/Tutorials`	Contains 6 Tutorial subfolders.
`Example Files/AnomalousInputExample.txt`	Anomalous Input File. See Chap. 3.4.
`Example Files/READ_ME.txt`	Explains how to use the examples.
`LuaScripts/`	Contains 4 examples of Lua scripts
`D+ Manual.pdf`	The User's Manual of D+

Table 1.2: The installed files include both files for a local installation, as well as a remote (Windows) installation. Depending on the options chosen at installation, you may only see a subset of these files.

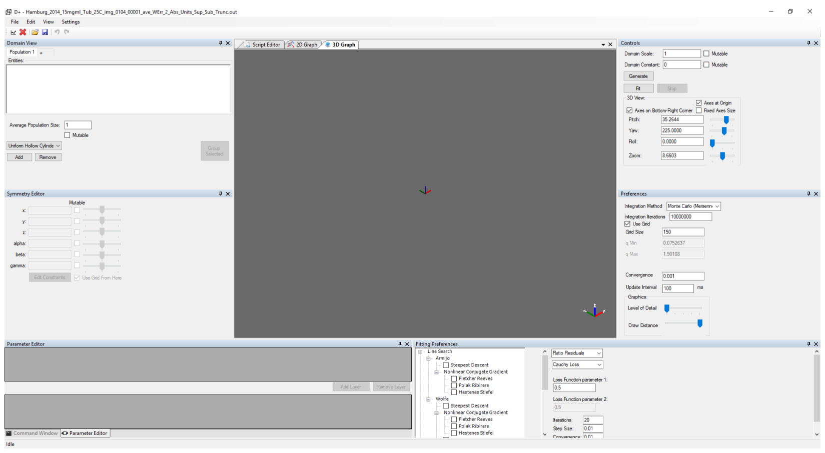
Chapter 2

Getting started

The secret of getting ahead is getting started.

Mark Twain (or not)

D+ computes the X-ray scattering from large supramolecular structures in solutions with high resolution. Both geometric and atomic models can be computed and combined in hierarchical manner to form complex structures. D+ can be launched either in remote mode or in local mode. By default, D+ launches in local mode. To launch D+ in the remote mode, run `DPlus.exe --remote`. This can be simplified by creating a shortcut to `DPlus.exe` and adding `--remote` to the target or by creating a batch script doing the same. If installed with the installer, there will be a `D+ Remote` shortcut in the Windows Start Menu. Once everything is installed correctly, open D+. Two windows should open. One looks like a command prompt, the other should look something like Figure 2.1.



D+ - Hamburg_2014_15mgml_Tub_25C_img_0104_00001_ave_WErr_2_Abs_Units_Sup_Sub_Trunc.out

Average Population Size: 1 ☐ Mutable

Symmetry Editor: x: y: z: alpha: beta: gamma: ☐ Mutable; ☑ Use Grid From Here

Domain Scale: 1 ☐ Mutable
Domain Constant: 0 ☐ Mutable
3D View: ☑ Axes at Origin ☑ Axes on Bottom-Right Corner ☐ Fixed Axes Size
Pitch: 35.2644
Yaw: 225.0000
Roll: 0.0000
Zoom: 8.6603

Integration Method: Monte Carlo (Mersenn
Integration Iterations: 10000000
☑ Use Grid
Grid Size: 150
q Min: 0.0752637
q Max: 1.90108
Convergence: 0.001
Update Interval: 100 ms

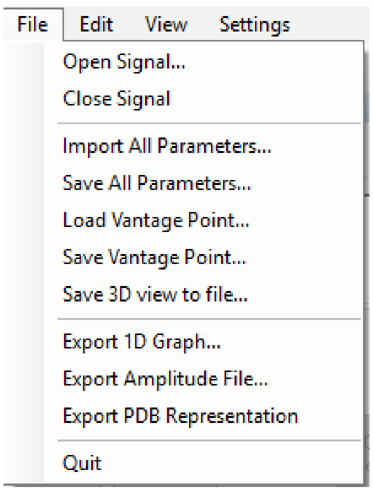
Ratio Residuals
Cauchy Loss
Loss Function parameter 1: 0.5
Loss Function parameter 2: 0.5
Iterations: 20
Step Size: 0.01
Convergence: 0.01

Idle

Figure 2.1: The main window contains multiple "dockable" tabs that can be placed based on personal preference. Each tab is addressed in this manual.

We shall address each of the panes here covering the various elements. The work-flow in D+ starts by defining a structure, continues by selecting the computation parameters, and ends by performing the computation itself. To help new users to get started, the folder of D+ has a `./Tutorials/` directory containing six tutorials, which cover the most important elements of D+. The `./Example Files/` directory includes several examples for the collective use of D+, some of which can be found in chapter 8. The ./LuaScripts/ directory has several examples of Lua scripts, some of which can be found in chapter 5.

2.1 Main window



There are a few functions that can be accessed from the main window. Under the File menu, Open Signal... and Close Signal load and close an experimental signal against which the model will be plotted, compared with, and fit to. Save All Parameters... saves all the parameters from the various panes (such as the Domain View, Parameter Editor, etc., but not from the Script Editor) to a file. The Script Editor has a separate save button. Import All Parameters... then loads all the parameters from that file and updates the graphic user interface (GUI). Save Vantage Point... saves all the parameters from 3D view, (sec. 2.9) Load Vantage Point... then loads all the parameters from that file and updates the GUI.

The option Save 3D view to file... is not implemented. Export 1D Graph... saves the calculated curve that is displayed in the 2D Graph pane. Export Amplitude File... and Export PDB Representation will be addressed in chapter 8.2. Quit exits D+ (surprise!).

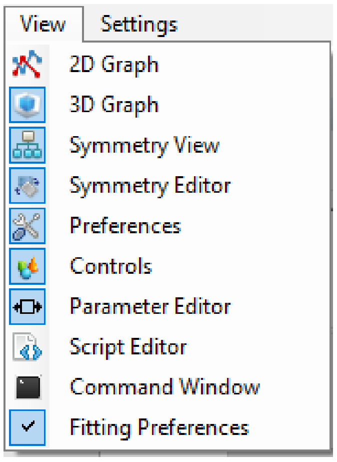
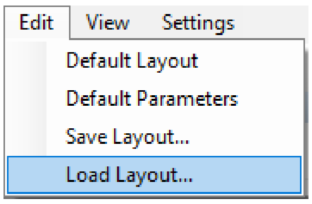
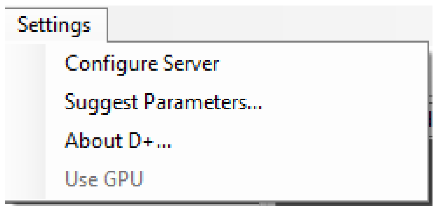


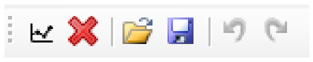
Figure 2.1 shows a possible layout of the main window. Change the layout based on personal preference. If a particular window or pane is closed, it can be reopened by going to the View menu and selecting the closed pane. This also brings panes that are tabs behind other tabs to the forefront of the other tabs. If you are looking for a pane that seems to have disappeared, this is a good place to help find it.

To save a particular layout, go to the Edit menu and select Save Layout.... A layout can be loaded by selecting Load Layout.... When closing D+, a file named Latest.dlayout will be created in the directory (or in %appdata%) so that the layout is preserved from session to session. The Default Layout is just a "common ground" layout and not a particularly comfortable one. Create one that feels right for you and save your own layout. Similarly, there is also an option to restore the Default Parameters in Controls, Preferences, and Fitting Preferences panes.





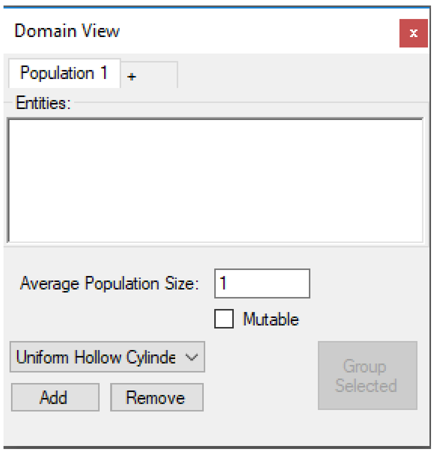
Other elements of the main window include the `Settings` menu and a status strip on the bottom of the window. The `Configure Server` option allows you to specify the server address and activation code. `Suggest Parameters...` opens the Suggest Parameters tool that suggests computational parameters based on the dimensions of the computed object. `About D+...` provide information about the current version of D+. When the computer has a GPU card, `Use GPU` is active and enables the user to choose whether to use it or not. The status strip at the bottom of the main opening window (Figure 2.1) should just have the word `Idle` on the left, as there is nothing happening when opening D+. Later when using D+, there may be different statuses as well as a progress bar.



In addition to the menus, there is a row of buttons near the top of the main window. The left four (`Open Signal...`, `Close Signal`, `Open Parameter File...` and `Save Parameters`) correspond to menu items in the `File` menu. The `Undo` and `Redo` buttons are unimplemented.

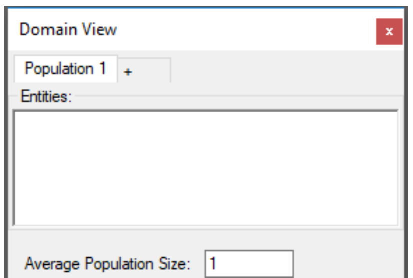
2.2 Domain View

The `Domain View` window contains the list of models that will be used to calculate a solution Small Angle X-Ray Scattering (SAXS) curve. We shall go over each element here.



Average Population Size: 1
Uniform Hollow Cylinde

First, you will note that on the top, there is a tab called `Population 1`. Multiple populations can be used. Different populations are assumed to have no correlations between them and therefore the scattering intensities (rather than amplitudes) are summed. Within each population there are correlations between the positions and orientations of subunits and therefore the scattering amplitudes of subunits are added. For explanations regarding the difference, see the published literature Ben-Nun et al. (2010), Als-Nielsen and McMorrow (2011), and Kittel (2005). Each population has an `Average Population Size` (default is 1). All the population sizes are given their respective weights, w_i, and the sum is divided by the total. In other words, the fraction of each population is given by $^{w_i}/_{\sum w_i}$. To add a population, press the `+` button to the right of the tab(s). Populations can be renamed for convenience by right clicking on the population tab and choosing `Rename` or `F2` (the `F2` option requires using the mouse before being able to press `F2`). Populations can be deleted by center clicking on the relevant tab or choosing `Close Population` from the context menu. The Mutable checkbox refers to the `Average Population Size`.



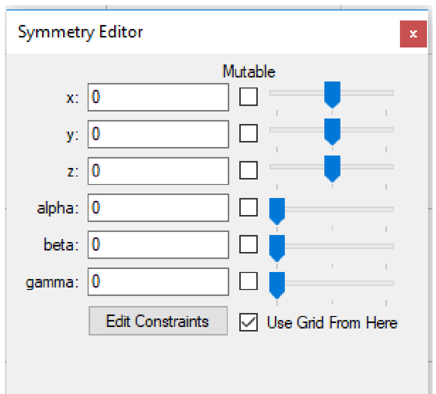
Average Population Size: 1

To add models to the population, select a model from the drop down menu on the lower left and press the `Add` button below it. Selected models can be renamed by right clicking on the model and choosing `Rename` or `F2`. The models are divided into three categories: geometric models; symmetries; and other. Other includes loading a `PDB file`, and loading a precalculated `Amplitude Grid (AMP)` from a file. To avoid incompatibility issues, when a precalculated amplitude is loaded, `q Max` and/or `Grid Size`, based on which the loaded amplitude was created, should not be further changed at the GUI. The grid is essentially a lookup table of amplitudes in the 3D reciprocal space ($\vec{q}$) used by D+ to calculate the scattering amplitude (and then intensity) of larger and more complex structures. Precise descriptions of the geometric models can be found in the paper "Solution X-ray scattering form factors of supramolecular self-assembled structures" Székely et al. (2010). The available geometric models are `Uniform Hollow Cylinder, Sphere, Symmetric Layered Slabs, Asymmetric Layered Slabs`, and `Helix`. After selecting a model and pressing `Add`, the model is included in the tree. Selecting the model provide access to `Parameter Editor`, in which the the electron density and dimensions can be provided. Additional layers of the same geometry can be added by pressing `Add Layer`. This bottom is in active in the `Helix` model. Layers can be added by using "*Symmetries*". The "*Symmetries*" will be addressed in full in chapter 4, but for now, just note that they must "wrap" other models. If one or more models are selected in the Entities box (from here on, the entity tree) and a `Symmetry` is selected in the drop down menu, the `Group Selected` button becomes active. Pressing it will "wrap" the selected model(s) in a new `Symmetry`. Models in the entity tree can be moved around by dragging and dropping. To remove a model, select it and press the `Remove` button or the delete key.

When the drop-down box is on `PDB File` or `Amplitude Grid (AMP)`, a `Center PDB` checkbox appears. When loading a PDB with the box checked, the atomic coordinates are translated such that the center of mass coincides with the origin $(0, 0, 0)$. Centering the structure around the origin is important since it minimizes the number of grid points, G, and hence the `Grid Size`, which are needed to satisfy the Nyquist-Shannon sampling rate (see more detail in section 2.10). Try it. Download a PDB file from the Protein Data Bank. Load the file with and without the checkbox checked. For an Amplitude File, the checkbox does not change anything except indicating that the box was checked in the headers of the resulting outputs.

2.3 Symmetry Editor

When a single item (or model) is selected in the entity tree, the `Symmetry Editor` window is activated. The x, y, z fields are the items location relative to the origin. The `alpha`, `beta`, and `gamma` fields dictate the orientation of the item, according to the convention of Tait - Bryan. Starting from `gamma`, which dictates the rotation about the z axis, followed



x: 0 ☐
y: 0 ☐
z: 0 ☐
alpha: 0 ☐
beta: 0 ☐
gamma: 0 ☐
✓ Use Grid From Here

by `beta`, which dictates the rotation about the y-axis and ending with `alpha`, which dictates the rotation about the x-axis. The rotation matrix is therefore:

$$\mathbf{A}(\alpha,\beta,\gamma) = \mathbf{A_x}(\alpha)\cdot\mathbf{A_y}(\beta)\cdot\mathbf{A_z}(\gamma) =$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \cdot \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta \\ \cos\alpha\sin\gamma+\cos\gamma\sin\alpha\sin\beta & \cos\alpha\cos\gamma-\sin\alpha\sin\beta\sin\gamma & -\cos\beta\sin\alpha \\ \sin\alpha\sin\gamma-\cos\alpha\cos\gamma\sin\beta & \cos\gamma\sin\alpha+\cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta \end{bmatrix}$$

The `Edit Constraints` button opens up a window allowing constraints to be set (only the absolute minimum/maximum are implemented). The `Use Grid From Here` checkbox is for “Hybrid” calculations and will be explained in more detail in section 7.1.

2.4 Command Window

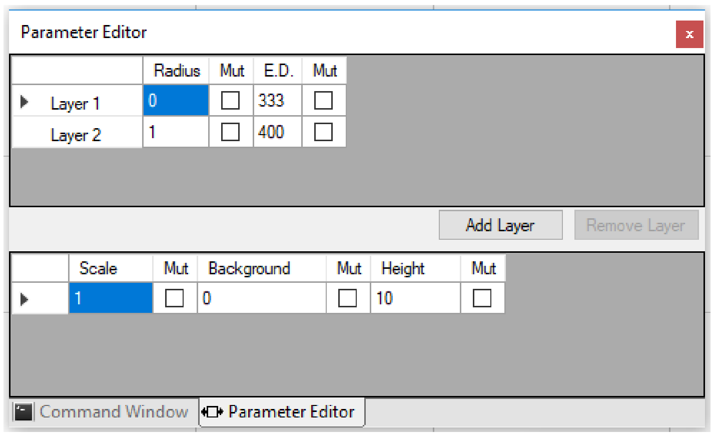
The `Command Window` provides access to some of the inner workings of D+. It can also be used as a calculator. The language used is `Lua`. For example, typing **math.pi**ˆ2 at the prompt gives ans = 9.86960440108936. Everything that can be written in a script can be entered here. See chapter 5 for details.

2.5 Script Editor

This pane is a basic `Lua` script editor. There are buttons to `Open`, `Save`, `Cut`, `Paste`, etc. The main advantage of this pane is the ability to run, pause and stop scripts. Breakpoints can also be set by using the `Add/Remove Breakpoint` button or by clicking next to the line numbers. See chapter 5 for details on scripting.

2.6 Parameter Editor

The `Parameter Editor` is where the parameters of each node in the entity tree can be edited. When one entity is selected, its parameters appear here (Figure 2.2). The upper panel contains a table of the model parameters. These parameters are often organized by layers, with the number of layers being variable. The bottom pane contains the *Extra Parameters* which is a fixed number of parameters characteristic of each model.

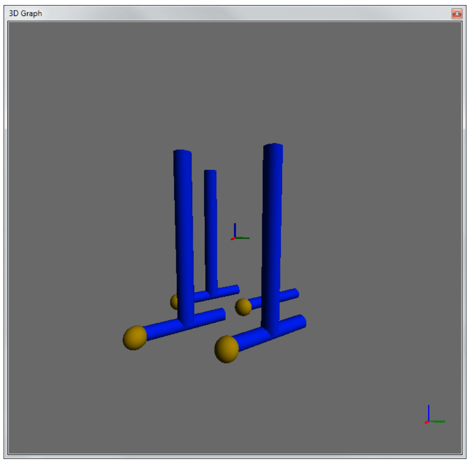


	Radius	Mut	E.D.	Mut
Layer 1	0	☐	333	☐
Layer 2	1	☐	400	☐

	Scale	Mut	Background	Mut	Height	Mut
	1	☐	0	☐	10	☐

Figure 2.2: `Parameter Editor` pane. Upper panel contains the model parameters of each layer (e.g., size (`Radius` in this example) or electron density (`E.D.`)). The lower pane contains extra global parameters that apply to all the layers.

2.7 3D Graph



The 3D Graph pane shows the real-space (or "real-world") picture whose solution X-ray scattering curve will be calculated. It can be rotated by holding down the right mouse button and dragging. Zoom is controlled by the scroll wheel. Items can be selected using the left mouse button. Dragging with the center mouse button changes the point upon which the camera focuses. Double clicking the center button returns the viewport to a semi-default setup. The camera can also be controlled from the `Controls` pane (sec. 2.9). Please note that if you have a large structure at atomic resolution it will take a long time to plot the 3D structure. It is better to work at minimal level of details and try to avoid the 3D Graph pane as much as possible.

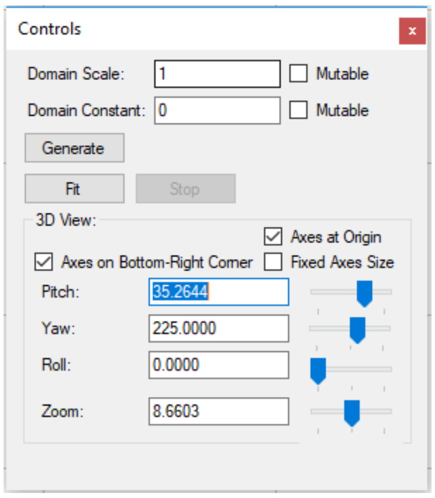
2.8 2D Graph

This window displays the calculated 2D graph of $I\left(q\right)$, which is the scattering intensity, I, as a function of the magnitude of the scattering vector, q, of the model described in `Domain View`. The model is displayed in blue, while a loaded signal is red. A loaded signal is usually the signal that we are trying to fit. Note that you might want to check the `Log(i)` and/or `Log(q)` boxes for viewing the curves. You can zoom in by clicking on the left mouse button twice to define a rectangle you want to view.

2.9 Controls

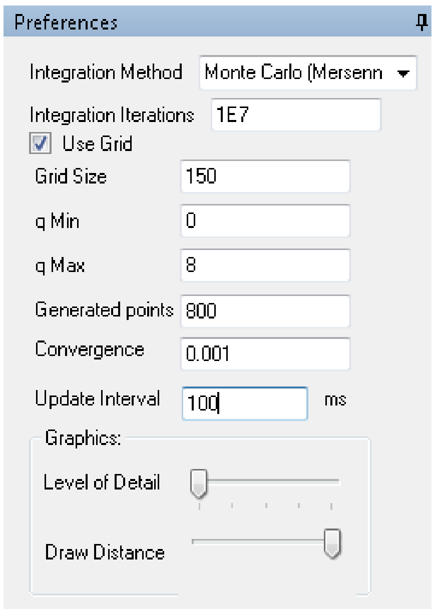
The `Controls` pane contains two main parts. The lower part, labeled `3D View`, controls the camera for the 3D Graph (sec. 2.7). There are the `Pitch`, `Yaw`, `Roll` and parameters as well as `Zoom`. Three checkboxes, `Axes at Origin`, `Axes on Bottom-Right Corner`, and `Fixed Axes Size` control the display of the reference red, blue and green axes in the 3D Graph.

The second (upper) part controls the generation and fitting of models. The domain scale (and its mutability checkbox) is a scale that the entire model is multiplied by. The Domain constant (and its mutability checkbox) is a constant that is added to the entire model. The `Generate` button starts the calculation of the current model. `Fit` will attempt to change the mutable parameters such that the model will better fit the loaded signal. The `Stop` button signals the calculation to abort. Note that stopping may not be immediate, (in particular when using GPU).



Domain Scale: 1 [ ] Mutable
Domain Constant: 0 [ ] Mutable
[✓] Axes at Origin
[✓] Axes on Bottom-Right Corner [ ] Fixed Axes Size
Pitch: 35.2644
Yaw: 225.0000
Roll: 0.0000
Zoom: 8.6603

2.10 Preferences



Preferences

Integration Method: Monte Carlo (Mersenn

Integration Iterations: 1E7

[✓] Use Grid

Grid Size: 150

q Min: 0

q Max: 8

Generated points: 800

Convergence: 0.001

Update Interval: 100 ms

Graphics:

Level of Detail

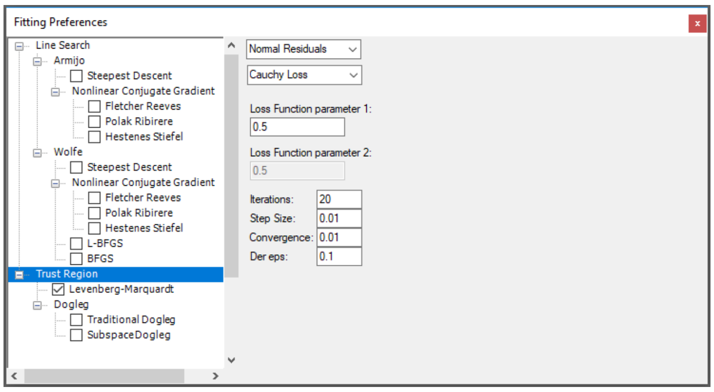
Draw Distance

The `Preferences` pane controls several parameters regarding the calculation of the modeled solution X-ray scattering curve. The `Integration Method`, `Integration Iterations`, and `Convergence` fields control how D+ calculates the orientation average of the model (more on this in chapter 6). The `Use Grid` checkbox determines whether or not lookup tables are allowed to be used in the calculations. Unchecking it may cause certain selection combinations not to work (`Adaptive (VEGAS) Monte Carlo`, for example).

`q Max` and `q Min` are the greatest and lowest q values for which the scattering intensity curve of the model will be calculated. Only when no signal is loaded, `q Max` and `q Min` can be modified. If a signal is loaded the `2D Graph` pane will show a curve. If there is a loaded signal and `q Max` and/or `q Min` should be change, the signal should be removed by the `Close Signal` option in the `File` menu or by pressing the thick red `X` button near the top of the main window. The default value of `q Min` is 0. In the current version, changing `q Min` does not affect the grid, which still contains all the points from 0 to `q Max`, hence the `Grid Size` parameter (which can be calculated by the `Suggests Parameters` tool, explained in sec. 2.12), should be computed assuming $q_{\min} = 0$. Orientation average is computed only between $q_{\min}$ and $q_{\max}$. `Generated Points (resSteps)` determines the number of points on the q-axis that will be generated (note that in the `state file`, which saves the GUI parameters, this variable is called `resSteps`). 800 generated points is the default and D+ reverts back to 800 when reopened (even when loading a saved state). Note that `q Min`, `q Max`, and `Generated Points (resSteps)` parameters will be hidden when a signal is loaded as they will be determined based on the q-axis of the loaded signal.

At the bottom of the pane there are two controls that affect the 3D Graph pane, `Draw Distance` and `Level of Detail`. If the `Level of Detail` is too high, rendering the scene (i.e, generating the image of the real-space model) will take a long time.

2.11 Fitting Preferences



Fitting Preferences

- Line Search
  - Armijo
    - [ ] Steepest Descent
    - Nonlinear Conjugate Gradient
      - [ ] Fletcher Reeves
      - [ ] Polak Ribirere
      - [ ] Hestenes Stiefel
  - Wolfe
    - [ ] Steepest Descent
    - Nonlinear Conjugate Gradient
      - [ ] Fletcher Reeves
      - [ ] Polak Ribirere
      - [ ] Hestenes Stiefel
    - [ ] L-BFGS
    - [ ] BFGS
- Trust Region
  - [✓] Levenberg-Marquardt
  - Dogleg
    - [ ] Traditional Dogleg
    - [ ] SubspaceDogleg

Normal Residuals

Cauchy Loss

Loss Function parameter 1: 0.5

Loss Function parameter 2: 0.5

Iterations: 20

Step Size: 0.01

Convergence: 0.01

Der eps: 0.1

D+ uses `Ceres-Solver` to try to fit mutable parameters to the data signal. This pane allows access to the various methods Ceres-Solver uses to solve the non-linear least-squares problem:

$$\begin{aligned}\min_{\mathbf{x}} \quad & \frac{1}{2}\sum_i \rho_i\left(\|f_i\left(x_{i_1},...,x_{i_k}\right)\|^2\right)\\ \text{Subject to:} \quad & l_j \le x_j \le u_j\end{aligned}$$

where x_i are the mutable variables. In our
case, the sum over i is just a single term or Residual Block in Ceres-Solver terminology. $f_i(\cdot)$ is a Cost-Function that produces the residuals of the objective function (and calculates the Jacobian when asked).

The initial input includes the experimental scattering intensity $I_{\text{experimental}}\left(q_i\right)$, an object that computes the expected scattering intensity $I_{\text{model}}\left(q_i\right)$, and a functor that evaluates the residuals. The residuals, f_i, can either be the `Normal Residuals`, $I_{\text{experimental}} - I_{\text{model}}$, a `Ratio Residuals`

$$1 - \left(\frac{I_{\text{experimental}}}{I_{\text{model}}}\right)^{\pm 1},$$

where the $\pm$ is chosen so that the residual is non-negative, or a `Logarithmic Residual`,

$$\left|\log\left(\frac{I_{\text{experimental}}}{I_{\text{model}}}\right)\right|.$$

$\rho_i\left(\cdot\right)$ is a Loss-Function that can be chosen by the user. The complete standard set of Loss-Functions (`Trivial Loss`, $\rho(s) = s$; `Huber Loss`, $\rho(s) = s$ if $s \le 1$ and $\rho(s) = 2\sqrt{s} - 1$ if $s > 1$; `Soft L1 Loss`, $\rho(s) = 2\left(\sqrt{1+s} - 1\right)$; `Cauchy Loss`, $\rho(s) = \log\left(s+1\right)$, etc.) from Ceres-Solver, is available in the GUI of D+. Cauchy, for example, provides a robust curve fitting as it can deal with outliers in the data (see `Ceres-Solver`). The method with which Ceres-Solver tries to fit to the data can also be chosen from several methods including `BFGS`, `LBFGS`, `Levenberg-Marquardt`, or `Dogleg`. Upper and lower bound constraints for each mutable fitting parameter are also supported by Ceres-Solver. Place the mouse cursor on a mutable parameter and right click on the mouse to obtain a new window in which the bounds can be added.

The (black) `Command Prompt` window will display messages when problems occur. The combination of `Lose-` and `Cost-Function` parameters dictate how the fitting will be performed when pressing the Fit button in the `Controls` pane.

The variables that control the fitting algorithms are `Step Size` that sets a limit on the fraction by which mutable parameters can be changed, `Iterations` - the maximum number of fitting attempts, `Convergence` - the cost function cutoff standard, and `Der eps`, which regulates the value of `Step Size` based on the change in the value of the cost function. The values of these variable should be adjusted and optimized for each case. Examples are provided in the paper of D+ and its supporting information.

D+ can find the *local* minimum for a plausible set of initial guess model parameters. Finding the initial guess should be done by several `Generate` iterations. Alternatively, one can use the `python API` of D+ to generate the scattering curves and do the optimization or fitting using the available fitting algorithms of Python, which may also include *global* fitting algorithms.

Fitting with certain options (for example, the thickness of a solvation layer) can take a significant amount of time (hours). When this is the case, it is better to use educated guesses. For small solvated structures (like soluble proteins), fitting with CRYSOL of FoXS is faster than D+ as no grids are computed in those programs.

To better know how to use the various combination of options and parameters see, for example, Ceres-Solver.

2.12 Suggest Parameters

D+ has a `Suggest Parameters` tool under the `Settings` menu, located at the main window. The tool gets as an input, `q Max`, and the `x`, `y`, and `z` coordinates of the point, P, which is most distant from the origin in the structure, for which grids are going to be used. If grids are used for the entire structure, P should be the most distant point in the entire structure. However, when the Hybrid method is used, P should be the most distant point in the part of the structure for which grids are used.

The tool computes the distance, L, of the most distant point from the origin. Practically, L is the higher of the following two value: the longest dimension in the structure (if the structure is centered around the origin - twice the radius of the sphere including the structure), or the most distant point from the origin in the structure. The `Suggest Parameters` tool can then compute, using Eq. 2.1, the suggested `Grid Size` for computations that use grid.

The number of points in the 3D grid, G (note the G is not `Grid Size`), is a measure of how dense the reciprocal-space grid (or lookup table) will be. G should be calculated according to the length of the structure L, contributing the highest frequency oscillation to the amplitude:

$$G \approx \left[(2q_{\text{max}})^3 - (2q_{\text{min}})^3\right] \cdot \left(\frac{L}{2\pi}\right)^3 .$$

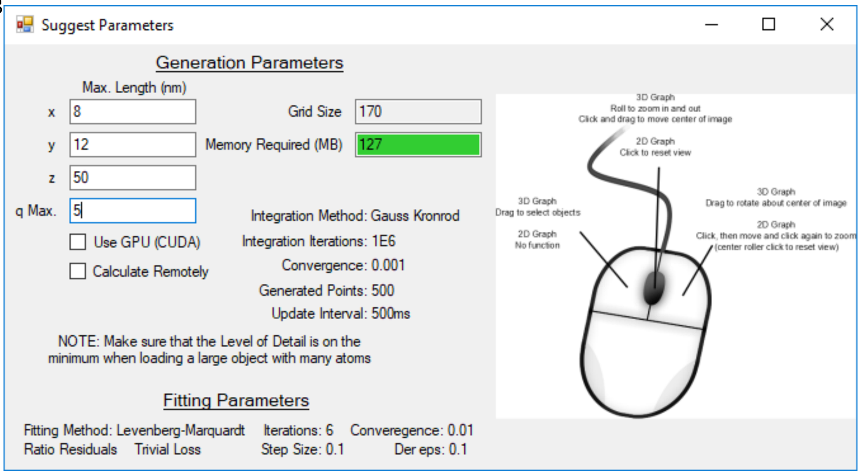
q_{max} and q_{min} are the maximum and the minimum magnitudes of the scattering vector, respectively. In the current version of D+ the number of grid points is calculated assuming $q_{\text{min}} = 0$, even when this is not the case. Note that this Nyquist-Shanon sampling rate is a rule of thumb, sometimes a denser grid is required. Further note that it is faster to compute structures (simple or complex) when their center of mass is at the origin, as smaller grid size will be required.

Using `q Max`, and the `x`, `y`, and `z` coordinates of the most distant point in the structure (with respect to the origin), or the longest dimensions of the structure in `x`, `y`, `z` (the larger of the two options), `Suggest Parameters` tool returns the `Grid Size` parameter, needed for D+, by computing:

$$\texttt{Grid Size} = 2N = \left(\left\lceil \frac{(q_{\text{max}} - q_{\text{min}}) \cdot L + 3}{10} \right\rceil + 1\right) \cdot 10. \qquad (2.1)$$

where N is the number of spherical grid shells. Therefore, there will be N points between the origin (or $q = 0$) and `q Max`, not including the origin. The reciprocal space grids have both negative and positive values of $\vec{q}$. The `Grid Size` parameter in D+ should therefore be an even number. If the `Grid Size` is an odd number, an error message is sent, in which the user is asked to modify the `Grid Size` to an even number. Equation 2.1 returns an integer number, which is divisible by 10.

The `Suggest Parameters` tool also returns the RAM volume (in units of MB) that is required for storing the `Grid`.



Suggest Parameters

Generation Parameters

Max. Length (nm)

x 8

y 12

z 50

q Max. 5

Grid Size 170

Memory Required (MB) 127

☐ Use GPU (CUDA)

☐ Calculate Remotely

Integration Method: Gauss Kronrod

Integration Iterations: 1E6

Convergence: 0.001

Generated Points: 500

Update Interval: 500ms

NOTE: Make sure that the Level of Detail is on the minimum when loading a large object with many atoms

Fitting Parameters

Fitting Method: Levenberg-Marquardt Iterations: 6 Converegence: 0.01

Ratio Residuals Trivial Loss Step Size: 0.1 Der eps: 0.1

3D Graph
Roll to zoom in and out
Click and drag to move center of image
2D Graph
Click to reset view
3D Graph
Drag to select objects
2D Graph
No function
3D Graph
Drag to rotate about center of image
2D Graph
Click, then move and click again to zoom
(center roller click to reset view)

It also suggests the suitable orientation average `Integration Method`, based on the aspect ratio of the structure and if GPU or CPU are used. Adaptive integration methods (`Gauss Kronrod` for CPU or `Vegas Monte Carlo` for GPU) are more suitable when the structure has a high aspect ratio.

The tool also suggests the number of `Integration Iterations` that should be used in D+, the `Convergence` cutoff, the number of `Generated Points` within the required q range, and the `Update Interval`, which the time in units of `ms` between successive computation updates.

If the diameter of the sphere that envelopes the structure is larger than L, the `x`, `y`, and `z` coordinates should be such that the resulting L will be that diameter. In that case, the integration parameters may not be optimal.

When the structure in not centered around the origin these suggestions are inaccurate, more details can be found in chapter 6). In any event, the suggested parameters are only a guide or first approximation. In practice, higher or lower values might be more appropriate for optimal results.

At the bottom of the `Suggest Parameters` pane, fitting method and Fitting Parameters are also suggested. These include the maximum number of `Iterations`, the fitting `Convergence` cutoff, the *cost* and *loss* functions, their maximum `Step Size` between fitting iterations, and its regularization parameter `Der eps`.

On the right side of the pane, there are detailed explanations of how to use the mouse in D+ to control the presentation in both `2D` and `3D Graph` views.

2.13 PDBUnits

`PDBUnits` is an accessory tool that can be used to identify the orientation and translations (or `Assembly Symmetry`) of a repeating subunit (given in `Subunit PDB file` in a complex supramolecular structure (given in a `Large PDB file`). The program also gets the tolerance, given by the maximum root-mean-squared-displacement (`RMSD`) value, within which repeating subunits can be considered similar. The program reads the PDB file of the complete structure and finds all the instances of the subunit. If subunits are splitted at different location in a PDB file, the parts should be combined into complete subunits, before using the tool. For each instance the rotation and translation with respect to the original subunit are computed. The tool exports the `Assembly Symmetry` of the subunit as a docking list (DOL) text file, which is a required input for D+, to compute the entire structure using the RG algorithmGinsburg et al. 2016.

The `*.dol` text file contains a list of the center of mass coordinates, `x`, `y`, and `z`, and the TaitBryan rotation angles α, β, and γ about the corresponding axes, of each subunit instance, within the large PDB file.

The program asks for the following items:

- The complete structure as a `Large unit PDB file` path
- The `Subunit PDB file` path
- The `RMSD` value, which sets the maximum `RMSD` allowed difference between the given `Subunit PDB file` and any instance of the subunit, within the `Large unit PDB file`.
- The `Output DOL file path`, in which the list of translations and rotations of repeating subunit instances will be stored.

The program can also be used from the `Command Prompt`. The default file path is:
`C:\Program Files\D+\bin>`
Within the folding, typing `PDBUnits --help`
gives the command line format:

```
PDBUnits <Large unit PDB file> <subunit PDB file> <RMSD> <Output DOL file path>
```

2.14 The Dplus Python API

The D+ Python API allows using the D+ backend from Python, instead of the ordinary D+ application.

The Python API works on both Windows and Linux.

Installation

Installing the Python API is done using PIP:

```
pip install dplus-api
```

The API was tested with Python 3.5 and newer. More details are provided in a separate README file.

Chapter 3

PDB (Atomic) Objects

> My religion consists of a humble admiration of the illimitable superior spirit who reveals himself in the slight details we are able to perceive with our frail and feeble mind.
>
> *Albert Einstein*

Using an atomic description of a molecule gives a much higher resolution picture than using geometric models. For example, consider the tubulin hetero-dimer. It is comprised of an α and a β dimer, both roughly spheres. However, the SAXS pattern of a pair of spheres is significantly different than that of the $\alpha\beta$ dimer. Using atomic representations can lead to much improved models, especially at higher resolutions (or higher q).

3.1 PDB Files

PDB files can be obtained from multiple sources. When downloaded from the protein data bank, the files usually work as-is. PDB files from simulations often disobey the standard and have various extensions.

A PDB file is a text representation of a collection atoms. From a certain point, atoms are represented like the following example:

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM    145  N   VAL A  25      32.433  16.336  57.540  1.00 11.92      A1   N
```

D+ requires that there be at least 78 characters per `ATOM` or `HETATM` (which stands for hetero (i.e. different) atom) line, as the atom type is determined by positions 77-78 (PDB v2.0 and greater)[1]. D+ reads and stores all the information, but the minimal information needed is shown here between the ⌊ and ⌋ symbols. Note that the spacing is critical.

[1]If older versions are used or the line is less than 77 characters long, then D+ will attempt to determine the atom type from characters 13-14. No charge is assumed in this case. Note that `HG` will be interpreted as mercury and not as "hydrogen-gamma" even though that may be the intent coming out of GROMACS, for example. Other deviations from the standard PDB file format should be amended (or removed from the file) before loading the file into D+.

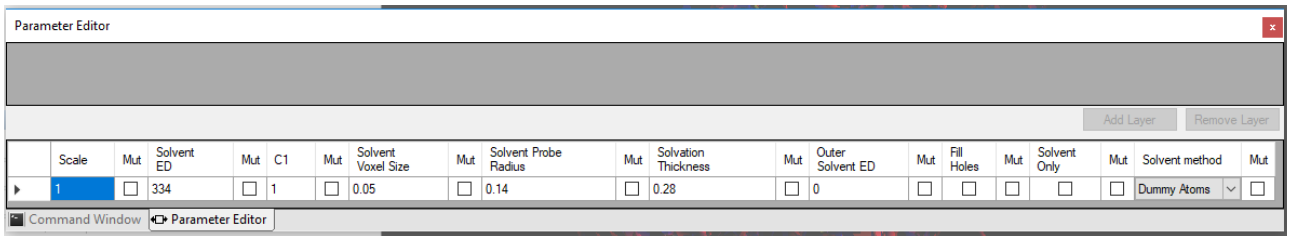
```
             1             2             3             4             5             6             7             8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM    145  N   VAL A  25      32.433  16.336  57.540  1.00 11.92          A1   N1+
⌊      ⌋                       ⌊      ⌋⌊      ⌋⌊      ⌋                         ⌊ ⌋
```

If there is a formal charge on the atom, it must appear in columns 79-80 of the line. A `HETATM` is processed exactly like an `ATOM`. Note that some simulations use various symbols to represent water molecules and these are not processed as such by D+. You will have to change the atom type to H and O manually.

MOLProbity **ChenDz5180** can be used to add hydrogen atoms to PDB files. PyMOL **PyMOL** can be used to add charges to residues that were supposed to be charged but the PDB file did not include the charges. The atomic form factor of ions, like O^-, differ from the form-factor of the corresponding neutral atom (O, in this example). D+ and take into account the atomic form factor of the 209 atoms and ions listed in the International Tables for Crystallography, using the five-Gaussian approximation **hamilton1974international**; **marsh1983corrections**. N^+, however, is not included in the International Tables for Crystallography and D+ computes the form factor of an N atom instead. If both hydrogen atoms and charges are added to an N^+ group, an excess electron is obtained. An approximation that might be used to better account for an N^+ ion, is to manually remove from the PDB file, one of the hydrogen atoms next to the N^+ ion. This ensures that the correct total number of electrons in the protein is preserved.

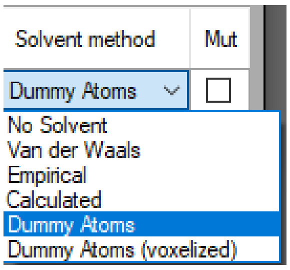
3.2 PDB Objects in D+

With high resolution data it is often useful to describe a protein structure as a collection of protein subunits. The subunits can often be resolved using other methods (crystallography, simulations, etc.) and their results are usually available as PDB files. One of the most important models D+ offers (accessible from the `Domain View`) is that of a PDB file. Once loaded, its parameters as shown in the `Parameter Editor` look like the following image.



	Scale	Mut	Solvent ED	Mut	C1	Mut	Solvent Voxel Size	Mut	Solvent Probe Radius	Mut	Solvation Thickness	Mut	Outer Solvent ED	Mut	Fill Holes	Mut	Solvent Only	Mut	Solvent method	Mut
▸	1	☐	334	☐	1	☐	0.05	☐	0.14	☐	0.28	☐	0	☐	☐	☐	☐	☐	Dummy Atoms	☐

Besides the `Scale` parameter, all the parameters relate to how the scattering amplitude of the solvent is subtracted from the atomic scattering amplitude. When considering geometric shapes (cylinders, spheres, etc.), the electron density *contrast* is trivial to obtain, as it is just the one shape minus a similar shape representing the solvent with a different electron density. In the case of a collection of atoms, treating the solvent is not so



trivial. D+ offers two main methods to subtract the solvent contribution, and account for the contribution of the solvation layer:

$$F(\vec{q}) = a \cdot F^{v}_{\text{mol}}(\vec{q}) - \rho_0 \cdot f^{\text{Dummy Atom}}_{\text{Excluded Solvent}}(\vec{q}) + (\rho_{\text{Solvation Layer}} - \rho_0) \cdot F_{\text{Solvation Layer}}(\vec{q}) \tag{3.1}$$

that uses `Dummy Atoms` Gaussian spheres to account for the excluded volume of atoms. Alternatively, the excluded volume can be taken into account as a collection of voxels:

$$F(\vec{q}) = a \cdot F^{v}_{\text{mol}}(\vec{q}) - \rho_0 \cdot F^{\text{Voxel}}_{\text{Excluded Solvent}}(\vec{q}) + (\rho_{\text{Solvation Layer}} - \rho_0) \cdot F_{\text{Solvation Layer}}(\vec{q}) \tag{3.2}$$

F^{v}_{mol} is explained in Equation 3.5, $f^{\text{Dummy Atom}}_{\text{Excluded Solvent}}$ in Equation 3.6, $F^{\text{Voxel}}_{\text{Excluded Solvent}}$ in Equation 3.8, and $F_{\text{Solvation Layer}}$ in Equation 3.10.

a is equal to 1, unless `Solvent Only` is indicated in the `Parameter Editor` of D+, in which case $a = 0$. If a is set to 0 and, in addition, `Solvent ED` (the mean solvent electron density, ρ_0), is set to 0, we get the contribution to the scattering amplitude only from the solvation shell. By loading a PDB and computing equation 3.1 using a finite value for the `Outer Solvent ED` (the solvation layer electron density, $\rho_{\text{Solvation Layer}}$), and `Solvent ED` (or ρ_0), in units of $e \cdot \text{nm}^{-3}$, the solvent contribution to the scattering is computed based on the radii taken from **fraser1978improved**; **slater1964atomic**; **svergun1995crysol**. The excluded volume and solvation shell are then calculated based on the chosen radii.

If `Outer Solvent ED`, ($\rho_{\text{Solvation Layer}}$), is set to 0, when a PDB file is loaded, D+ does not add any solvation layer. If the same PDB is reloaded and only the contribution from the solvation shell is computed [by indicating `Solvent Only` (i.e. $a = 0$), and by setting `Solvent ED`, ρ_0, to 0, and `Outer Solvent ED`, $\rho_{\text{Solvation Layer}}$, to $\rho_{\text{Solvation Layer}} - \rho_0$], any other method for calculating the solvation shell can be applied and its contribution can be added to the contributions of the vacuum term (i.e `Outer Solvent ED`=0, and `Solvent ED` = 0) and the `Dummy Atoms` term. Splitting the computation in this mode has another advantage. Fitting the value of the solvation layer contrast, $\rho_{\text{Solvation Layer}} - \rho_0$, becomes faster, as it is equivalent to setting `Ourter Solvent ED`=1 (or $\rho_{\text{Solvation Layer}} = 1$) and fitting the `Scale` factor of the solvation shell amplitude, and hence does not require to compute a new grid for the voxels of the solvation shells.

Gaussian "Dummy" Atoms

To account for the excluded volume of the atoms, it has been suggested to subtract a Gaussian sphere from each atom, where the center of the sphere is located at the atom coordinate **fraser1978improved**. Therefore, D+ subtracts the Fourier transform of the following normalized Gaussian sphere:

$$F^{d}_{j}(q) = \rho_0 \cdot V_j \cdot \exp\left[-\frac{V_m^{2/3} \cdot q^2}{4\pi}\right],$$

where $V_j = \frac{4\pi}{3} r_j^3$ is the volume of atom j, and $V_m = \sum_j V_j$ is the mean atomic volume over the entire molecule. r_j is an effective atomic radius (dictated by the type of atom) and ρ_0 is the mean solvent electron density (`Solvent ED`). D+ accepts both positive and negative values of ρ_0. Negative ρ_0 values are nonphysical, however, they might be useful in special cases. D+ always **subtracts** the excluded volume amplitude, F_j^d, from other amplitudes (see Equation 3.3). Negative ρ_0 can be used to **add** the contribution of F_j^d to other amplitudes. Caution should be taken, if solvation layer is added, as a negative ρ_0 value will also impact Equations 3.1, 3.2, 3.9, and 3.11. The scattering amplitude from atom j after subtracting the contribution of the solvent is:

$$f_j^\circ(q) - F_j^d(q) \tag{3.3}$$

where $f_j^\circ(q)$ is the scattering amplitude of the j-th atom *in vacuo* (using the five Gaussian approximation):

$$f_i^0(q) = \sum_{k=1}^{4} a_k \cdot \exp\left(-b_k\left(\frac{|\vec{q}|}{4\pi}\right)^2\right) + c. \tag{3.4}$$

The coefficients a_k, b_k, and c are the Cromer-Mann coefficients, given in units of the Thomson scattering length, r_0. Note that in D+, however, r_0 is set to 1.

The solvent contribution can be modulated like

$$f_j^s(q) = f_j^\circ(q) - C_1(q)\,F_j^d(q)$$

where

$$C_1(q) = c_1^3 \cdot \exp\left[-\frac{V_m^{2/3}\cdot q^2\left(c_1^2-1\right)}{4\pi}\right].$$

is used to uniformly adjust V_m and c_1 is a scaling factor (**schneidman2013accurate**). The default value for c_1 is 1 (making it not affect the solvent subtraction) and the default constraints are $[0.95, 1.05]$. This method is coined `Dummy Atoms` in the drop down menu above.

The solution scattering amplitude from a molecule, given by a list of n atoms, whose coordinates are $\vec{r}_j$, is:

$$F_{\text{mol}}^s(\vec{q}) = \sum_{j=1}^{n} f_j^s(q)\cdot\exp\left(i\vec{q}\vec{r}_j\right).$$

If `Solvent ED` or $\rho_0 = 0$, we get the scattering amplitude of the molecule in vacuum:

$$F_{\text{mol}}^v(\vec{q}) = \sum_{j=1}^{n} f_j^0(q)\cdot\exp\left(i\vec{q}\vec{r}_j\right). \tag{3.5}$$

The contribution of the excluded volume to the amplitude is

$$\rho_0 \cdot f_{\text{Excluded Solvent}}^{\text{Dummy Atom}} = C_1(q)\cdot\sum_{j=1}^{n} F_j^d(q)\cdot\exp\left(i\vec{q}\vec{r}_j\right) \tag{3.6}$$

All the other options for solvent subtraction pertain to the following method.

Voxel Based Solvent

An alternative solvent subtraction method is to determine the shape that is the union of all the atoms and subtract that (at the electron density of the solvent). This was proposed by **Pavlov**. This methods makes a few assumptions and has some shortcomings.

The solvent excluded volume is calculated in several stages. First of all, space is discretized into voxels of a specific size, determined by the `Solvent Voxel Size` parameter, given in nm and its default value is 0.2 nm. Each voxel is marked as occupied if at least one atom's coordinate is within that atom's radius (the atomic radius type can be chosen in the `Solvent method` parameter, including the radii of the Gaussian dummy atoms, given in `Dummy Atoms (voxelized)`). This represents the excluded volume of the atomic structure.

The contribution of the excluded solvent to the scattering amplitude is computed as the sum over the scattering amplitudes from the collection of voxels (or boxes) comprising the solvent excluded volume. The scattering amplitude of the jth voxel (or box) of dimensions, ω_j, τ_j, and μ_j, whose center is at $\vec{r}_j^{\text{Voxel}}$ is (see **szekely2010solution**):

$$f_j^{\text{Voxel}}\left(\vec{q}\right) = \frac{8}{q_x \cdot q_y \cdot q_z} \sin\left(\frac{q_x\omega_j}{2}\right) \sin\left(\frac{q_y\tau_j}{2}\right) \sin\left(\frac{q_z\mu_j}{2}\right) \cdot \exp\left(i \cdot \vec{q} \cdot \vec{r}_j^{\text{Voxel}}\right) \tag{3.7}$$

The total scattering amplitude of the excluded voxels is then a sum over the solvent excluded voxels:

$$F_{\text{Excluded Solvent}}^{\text{Voxel}}\left(\vec{q}\right) = \sum_{j \in \{\text{Excluded Voxels}\}} f_j^{\text{Voxel}}\left(\vec{q}\right) \tag{3.8}$$

and the scattering amplitude of the molecule in the solution is:

$$F_{\text{mol}}^{s}\left(\vec{q}\right) = F_{\text{mol}}^{v}\left(\vec{q}\right) - \rho_0 \cdot F_{\text{Excluded Solvent}}^{\text{Voxel}}\left(\vec{q}\right). \tag{3.9}$$

Solvation Layer

To obtain the contribution of a solvation layer around an atomic structure, the solvent excluded volume is determined, as explain in 3.2, using the selected `Solvent method`. A probe with a radius of `Solvent Probe Radius` (in nm) determines the accessible surface of the protein. The default value of the probe radius is 0.14 nm, which corresponds to the radius of a water molecule. All voxels that are within `Solvation Thickness` (in nm) of the surface and are not occupied by an atom are marked as occupied by a solvation layer, and their mean electron density (ED), $\rho_{\text{Solvent Layer}}$, is given by the value of `Outer Solvent ED` (in units of $e \cdot \text{nm}^{-3}$). If `Outer Solvent ED` $= 0$, D+ ignores the contribution from the solvation layer. Voxels that are completely enveloped by the union of the two shapes are considered holes. If the `Fill Holes` checkbox is checked, these voxels will be added to the shape that represents the volume of the atoms (as excluded volume). The excluded volume shape with an electron density of `Solvent ED`, ρ_0 (in units of $e \cdot \text{nm}^{-3}$), is subtracted from the atomic contributions (Equation 3.9). The solvation layer represents the electron density contrast owing to a higher or lower density of the solvent near the surface of the atomic structure, with respect to the bulk solvent.

The contribution of the solvation layer to the scattering amplitude is computed as the sum over the scattering amplitudes from the collection of voxels comprising the solvation layer. The amplitude of the solvation layer is then:

$$F_{\text{Solvation Layer}}\left(\vec{q}\right) = \sum_{j\in\{\text{Solvation Layer Voxels}\}} f_j^{\text{Voxel}}\left(\vec{q}\right). \tag{3.10}$$

The scattering amplitude of the solvated molecule is:

$$F_{\text{Solvated Molecule}}\left(\vec{q}\right) = F_{\text{mol}}^{s}\left(\vec{q}\right) + \left(\rho_{\text{Solvation Layer}} - \rho_0\right)\cdot F_{\text{Solvation Layer}}\left(\vec{q}\right). \tag{3.11}$$

Computations of voxel based excluded volume and solvation layer can take a significant amount of time, particularly when using CPU. Fitting the parameters should therefore be done manually rather than by the fitting algorithms of D+. In §9 of the Supporting Online Materials (SOM) of the paper about D+ **Dplus2017** there are several examples that demonstrate how to compute the solvation layer of large complexes, containing many subunits, in a scalable manner.

Implicit/Explicit Hydrogen Atoms

Hydrogen atoms are often not listed in PDB files that originate in protein and/or nucleic acid crystallography. On the flip side, they are usually included in files originating from simulations, or can be added to the PDB file by software like *MolProbity* (see **ChenDz5180**). Furthermore, their scattering contribution is not negligible in proteins and/or nucleic acids. To accurately take the hydrogen atoms into account when they are missing from the PDB file, each amino acid or nucleic base is looked at (based on the number at positions 23-26 in the PDB file). If there is at least one hydrogen atom within the amino acid residue or nucleic base, then nothing is changed. If there are no hydrogen atoms, then the collection of lines is further examined. If all the atoms correspond to atoms within the specific amino acid or nucleic base (based on characters 13-16, *e.g.* `CG2`, or carbon-gamma number 2) then the different atoms within the amino acids or nucleic bases are modified so that their relevant hydrogen atoms' contributions are added to the heavier atoms they are bonded to. If the amino acid or nucleic base name (characters 23-26) is not recognized, there is at least one explicit hydrogen atom in the amino acid or nucleic base, or if there is an unidentified atom (characters 18-20) then only the explicitly listed hydrogens are taken into account. Otherwise, they will be taken into account implicitly.

The implicit addition of H atoms modifies the five Gaussian approximation for the heavier atom (see Table 1 in **Dplus2017**). The implicit hydrogen atoms change the heavier atoms' effective radii as in Table 1 in **svergun1995crysol**. This affects the subtracted solvent and the radii used for the solvation layer.

3.3 PDB files of structures with repeating subunits

If a large PDB file contains repeating subunits, the computation can be done faster, using `PDBUnits`, which finds all the instances of the subunit within a large structure. The tool exports the `Assembly Symmetry` of the subunit as a docking list (DOL) text file, which can then be used by D+ together with the PDB of the subunit to compute the contribution of all its instances to the scattering amplitude.

3.4 Anomalous Scattering

D+ can take anomalous scattering of atoms into account when computing scattering amplitudes of atomic models. Two additional correction terms should be added to get the atomic form factor energy dependence:

$$f(q,\lambda) = f^{\circ}(q) + f'(\lambda) + if''(\lambda).$$

When selecting a `PDB File...` in the the dropdown menu in `Domain View` window, the `Anomalous` checkbox will appear. Selecting the checkbox will cause a file selection window to launch, and you can choose an `Anomalous file`. Unselecting the checkbox will cause the `Anomalous File` to be removed. Select the text file containing the relevant f' and f'' values. The file should be formated using any combination of the lines below.

```
# Energy    2345.82km
# <f'>  <f''>   <atom identifier>
1.2     2.3     Fe2+
#<f'>   <f''>   <atom indices>
1.3     3.4     125     256
#<f'>   <f''>   <name of a group that's defined below>
-3e4    +2.01   $(someRandomGroupName)
#<name of a group>      <list of atom indices>
$(someRandomGroupName)  252 472 7344 313
```

Lines that start with '#' are considered comments and are ignored. The `Example Files` subfolder of D+ includes more examples.

Please note: only **Monte Carlo** integration method can be used for anomalous scattering.

3.5 DebyeCalculator

`DebyeCalculator` is a command line executable program that is located at `C:\Program Files\D+\bin>` or where ever D+ was installed.

In this folder, typing `DebyeCalculator.exe -h` will print the help menu of the program, which provides the allowed options:

`-h [ --help ] arg`
Print this help message

`-i [ --PDBFile ] arg`

Input PDB file

`-o [ --out ] arg (=Debye.out)`

The name of the file to write the results to

`-g [ --gpu ] [=arg(=1)] (=0)`

Use a GPU to calculate if available

`-p [ --progress ] [=arg(=1)] (=1)`

Print progressbar

`-s [ --sol ] [=arg(=333)]`

The solvent electron density to be subtracted as Gaussians

`-c [ --c1 ] arg (=1)`

The c1 value used to adjust the excluded volume of the atoms

`-d [ --deb ] [=arg(=1)] (=0)`

Take the B factor into account if available

`--qMin arg (=0)`

The minimum q value from which the intensity will be calculated

`-q [ --qMax ] arg (=6)`

The maximum q value to which the intensity will be calculated

`-n [ --qVals ] arg (=512)`

The number of points to be calculated

`-k [ --kernel ] arg (=2)`

The version of the GPU kernel to use. Not safe. Don't assume any number works unless you know what you're doing. Valid options: are `1,2`, or `4`. Other values will fail.

Examples

The simplest usage example would be:

```
C:\Program Files\D+\bin>DebyeCalculator.exe -i C:\Users\owner\Downloads\1jff.
pdb -o C:\Users\owner\Downloads\1jffDebye.out
```

The same results will be obtain by:

```
C:\Program Files\D+\bin>DebyeCalculator.exe -i C:\Users\owner\Downloads\1jff.
pdb -o C:\Users\owner\Downloads\1jffDebyeS.out -g 0 -p 1 -s 333 -c 1 -d 0 0 -q 6 -n 512
```

Chapter 4

Symmetries

The universe is built on a plan the profound symmetry of which is somehow present in the inner structure of our intellect.

Paul Valery

4.1 Introduction

Supramolecular complex structures can be often described as hierarchical data trees in which repeating subunits are docked into their assembly symmetries. The assembly symmetries describe how repeating subunits are organized (in other words, translated and rotated) is space (see Figure 4.1). For complex assemblies, the scattering amplitude is given by:

$$F\left(\vec{q}\right) = \sum_{j=1}^{J}\sum_{m=1}^{M_j}\left[F_j\left(\mathbf{A}_{j,m}^{-1}\vec{q}\right)\cdot\exp\left(i\vec{q}\cdot\vec{R}_{j,m}\right)\right] \tag{4.1}$$

where $F\left(\vec{q}\right)$ is the scattering amplitude from an assembly comprised of n_s subunits. J is the number of different types of objects, which is also the total number of leaves in the hierarchical tree structure representation of the entire supramolecular assembly (Figure 4.1). The leaves can either be geometry-based or atomic-based models, taken, for example, from Protein Data Bank (PDB) files. M_j is the number of instances of object type j, whose orientations and positions are determined by rotation matrices $\mathbf{A}_{j,m}$ (see section 2.3) and real-space translation vectors $\vec{R}_{j,m}$, respectively. The total number of subunits, n_s, is therefore $\sum_{j=1}^{J} M_j$.

To evaluate the scattering of a complex structure, we need a way to describe “Assembly Symmetries”. In D+, there are three methods to do so.

4.2 Space-Filling Symmetry

Background

In this method, all child subunits are treated as a unit cell of a primitive Bravais lattice. Copies of unit cell object type j at the same orientation, given by matrix $\mathbf{A}_j$, are placed at $R_{j,m} =$

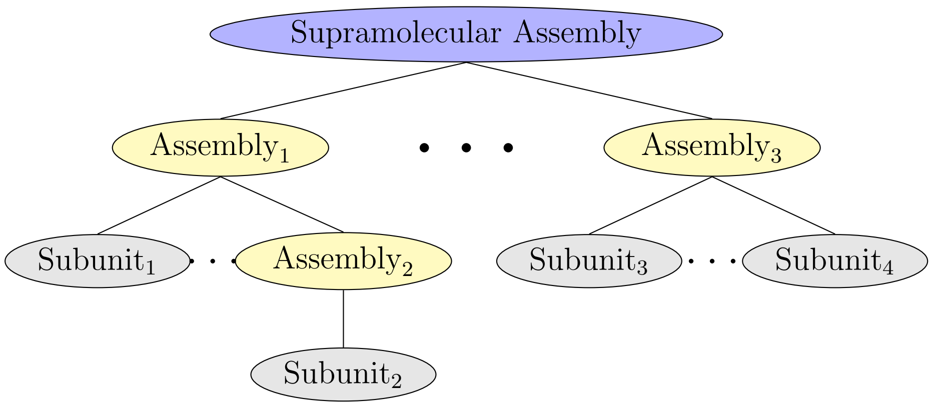


Figure 4.1: Example of modeling a supramolecular assembly in a hierarchical manner. Each *Assembly Symmetry* may contain multiple children. Children can either be additional *Assembly Symmetries* or *Subunits*. A *Subunit* represents an atomic model (for example, a Protein Data Bank (PDB) file) or a geometric model. Internal nodes consist of *Assembly Symmetries*, whereas each leaf must be a *Subunit*. There may be an arbitrary number of hierarchy levels and nodes in each level. Many supramolecular structures may be constructed that way. Care, however, should be taken when dealing with very large structures (see **Hybrid Calculations** for details).

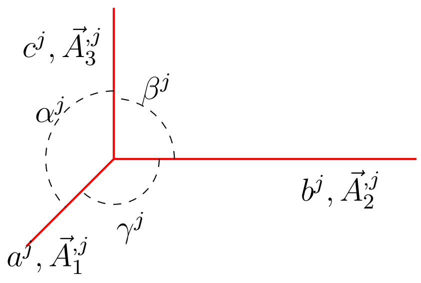


Figure 4.2: Unit cell vectors. a^j is the length of $\vec{A}_1^j$, which is laying along the x-axis, b^j is the length of $\vec{A}_2^j$, which is within the xy plane, and c^j is the length of $\vec{A}_3^j$. The angles between the unit cell vectors are indicated in the Figure.

$R^j_{n_1^j,n_2^j,n_3^j} = n_1^j \cdot \vec{A}_1^j + n_2^j \cdot \vec{A}_2^j + n_3^j \cdot \vec{A}_3^j$, where m is an index that corresponds to $\{n_1^j, n_2^j, n_3^j\}$, which is any combination of three integers between $\{0,0,0\}$ and $\{(N_1^j-1),(N_2^j-1),(N_3^j-1)\}$. The unit cell vectors, $\vec{A}_1^j, \vec{A}_2^j$, and $\vec{A}_3^j$ are defined by three lengths a^j, b^j, and c^j and three angles α^j, β^j, and γ^j (Figure 4.2). The total number of subunit copies, M_j, of object j will be $N_1^j \times N_2^j \times N_3^j$. The contribution of the space-filling lattice, comprise of repeating subunits of type j, to the scattering amplitude, F_J, is computed according to:

$$F_J(\vec{q}) = F_j\left(\mathbf{A}_j^{-1}\vec{q}\right) \cdot \exp\left(i\vec{q} \cdot \vec{T}_j\right) \cdot SF^j(\vec{q}), \tag{4.2}$$

where $\mathbf{A}_j$ is the rotation matrix by which subunit j was rotated, $\vec{T}_j$ is a translation vector of the entire space-filling lattice, and $SF^j(\vec{q})$ is the space-filling structure-factor, given by:

$$SF^j(\vec{q}) = \sum_{n_1^j=0}^{(N_1^j-1)} \sum_{n_2^j=0}^{(N_2^j-1)} \sum_{n_3^j=0}^{(N_3^j-1)} \exp\left(i\vec{q} \cdot \vec{R}^j_{n_1^j,n_2^j,n_3^j}\right). \tag{4.3}$$

The three unit vectors in real-space are constructed in the following way. We start with vector $\vec{A}_1^j$ of length a^j along the x direction and vector $\vec{A}_2^j$ of length b^j. The angle between the two vectors is γ^j. We then add a third vector $\vec{A}_3^j$ defined by its length c^j and two more angles α^j and β^j, where α^j is between the vectors $\vec{A}_1^j$ and $\vec{A}_3^j$, and β^j is between the vectors $\vec{A}_2^j$ and $\vec{A}_3^j$. In real-space, the basis vectors are given by:

$$\vec{A}_1^j = (a^j, 0, 0),$$

$$\vec{A}_2^j = \left(b^j \cos\gamma^j, b^j \sin\gamma^j, 0\right),$$

$$\vec{A}_3^j = \left(c^j \cos\alpha^j, \frac{c^j \cdot t^j}{\sin\gamma^j}, \frac{c^j \cdot B^j}{\sin\gamma^j}\right),$$

where $t^j = \cos\beta^j - \cos\alpha^j \cos\gamma^j$ and $B^j = \sqrt{\sin^2\gamma^j - \sin^2\gamma^j \cos^2\alpha^j - (t^j)^2}$.

The angles should satisfy the conditions that the sum of any pair of angles is larger (or equal) than the third angle and that $\sin\gamma^j \neq 0$ and $\sin^2\gamma^j - \sin^2\gamma^j\cos^2\alpha^j - (t^j)^2 > 0$. D+ checks that these conditions are satisfied. If the sum of two angles equals the third angle, or if $\sin\gamma^j = 0$, the symmetry is 2D. In this case, the correct usage of the D+ is to project the 2D symmetry on the x-y plane. The projection is done by providing the values of a^j, b^j, and γ^j, set both α^j and β^j to be 90, and set the number of repeating subunits in the third (z) direction to be 1.

In real-space, the unit cell vectors $\vec{A}_h$ can then be rotated by a rotation matrix $\mathbf{O}_j$, so that the final unit cell vectors are: $\vec{a}_h^j = \mathbf{O}_j \vec{A}_h^j$, where $h \in \{1, 2, 3\}$, and then $R_{j,m} = R^j_{n_1^j, n_2^j, n_3^j} = n_1^j \cdot \vec{a}_1^j + n_2^j \cdot \vec{a}_2^j + n_3^j \cdot \vec{a}_3^j$.

What can we expect to find in reciprocal-space?

As the basis vectors in reciprocal space are given by:

$$\vec{a}_1^{*j} = \frac{2\pi}{\upsilon^j} \cdot \left(\vec{a}_2^j \times \vec{a}_3^j\right), \; \vec{a}_2^{*j} = \frac{2\pi}{\upsilon^j} \cdot \left(\vec{a}_3^j \times \vec{a}_1^j\right), \; \vec{a}_3^{*j} = \frac{2\pi}{\upsilon^j} \cdot \left(\vec{a}_1^j \times \vec{a}_2^j\right)$$

where $\upsilon^j = \vec{a}_1^j \cdot \left(\vec{a}_2^j \times \vec{a}_3^j\right)$, we get:

$$\vec{a}_1^{*j} = \left(\frac{2\pi}{a^j}, \frac{-2\pi\cos\gamma^j}{a^j \sin\gamma^j}, \frac{2\pi\left(t^j \cdot \cos\gamma^j - \cos\alpha^j \sin^2\gamma^j\right)}{a^j \cdot B^j \sin\gamma^j}\right),$$

$$\vec{a}_2^{*j} = \left(0, \frac{2\pi}{b\sin\gamma^j}, \frac{-2\pi t^j}{b^j \cdot B^j \sin\gamma^j}\right),$$

$$\vec{a}_3^{*j} = \left(0, 0, \frac{2\pi\sin\gamma^j}{c^j \cdot B^j}\right).$$

Any $\vec{q}$-vector, which is a linear combination of the reciprocal-space basis-vector, and is given by

$$\vec{G}^j_{h_j,k_j,l_j} = h_j \vec{a}_1^{*j} + k_j \vec{a}_2^{*j} + l_j \vec{a}_3^{*j},$$

where h_j, k_j, and l_j are integers, satisfies Bragg's condition, hence contributes (according to equation 4.3) to a structure-factor correlation peak at $\vec{q} = \vec{G}^j_{h_j,k_j,l_j}$. In solution, the contribution

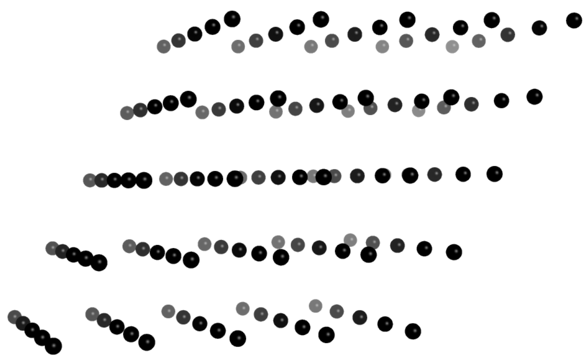


Figure 4.3: Example of a 3D hexagonal lattice.

of SF^j to the scattering curve will be at the scattering-vector amplitudes, q, which satisfy Bragg's condition, and are given by:

$$q = \left|\vec{G}^j_{h_j,k_j,l_j}\right| = \left|h_j\vec{a}^{*j}_1 + k_j\vec{a}^{*j}_2 + l_j\vec{a}^{*j}_3\right| =$$

$$\frac{2\pi}{a^j \cdot b^j \sin\gamma^j} \cdot \sqrt{\begin{array}{c}\left(h_j b^j \sin\gamma^j\right)^2 + \left(k^j \cdot a^j - h_j \cdot b^j \cos\gamma^j\right)^2 + \\ \frac{\left(h_j b^j c^j\left(\cos\gamma^j \cos\beta^j - \cos\alpha^j\right) - k_j a^j c^j\left(\cos\beta^j - \cos\alpha^j \cos\gamma^j\right) + l_j a^j b^j \sin^2\gamma^j\right)^2}{\left(c^j\right)^2\left(\sin^2\gamma^j - \cos^2\alpha^j - \cos^2\beta^j + 2\cos\beta^j \cos\alpha^j \cos\gamma^j\right)}\end{array}} .$$

The line-shape of the peaks will follow Equations 4.2 and 6.1.

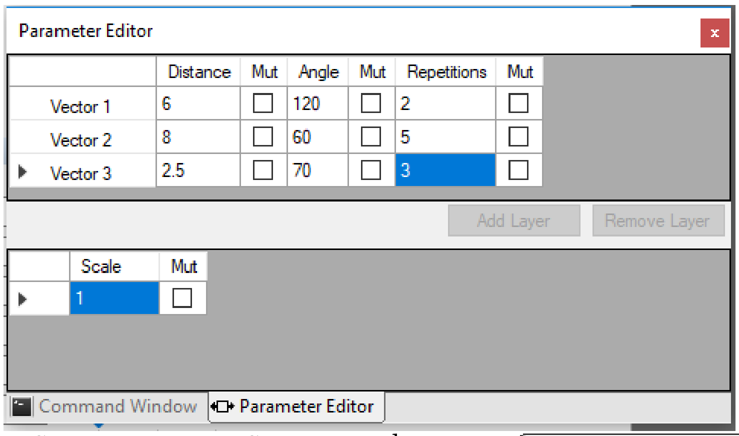
Space-filling is a relatively easy way to create a regular Bravais lattice. For example, a 3D hexagonal lattice is defined by:

$$\left\{a^j = b^j, c^j, \alpha^j = \beta^j = 90°, \gamma^j = 120°\right\}$$

Figure 4.3 shows an example of a 3D hexagonal lattice with five repeats in each direction.

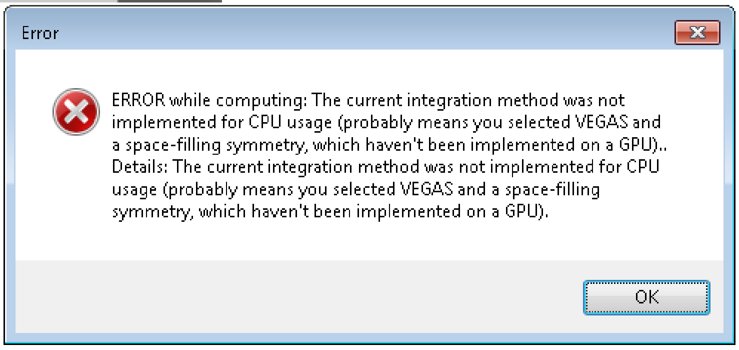
Using Space-Filling Symmetry in D+

To use space-filling symmetry in D+, go to `Domain View`, select `Space-filling Symmetry`, and press `Add`. Then select the subunit model to which the space-filling symmetry applies and drug it with the mouse into the relevant `Space-filling symmetry`. To edit the parameters of the space-filling symmetry, select it and edit the values in the `Parameter Editor` of the space-filling symmetry, which looks as shown here. The input parameters include the three lattice distances a^j, b^j, and c^j, the three lattice angles (in degrees) α^j, β^j, and γ^j, as well as the number of subunit `Repetitions` in each of the three lattice `Vectors`.


Parameter Editor

	Distance	Mut	Angle	Mut	Repetitions	Mut
Vector 1	6	☐	120	☐	2	☐
Vector 2	8	☐	60	☐	5	☐
Vector 3	2.5	☐	70	☐	3	☐

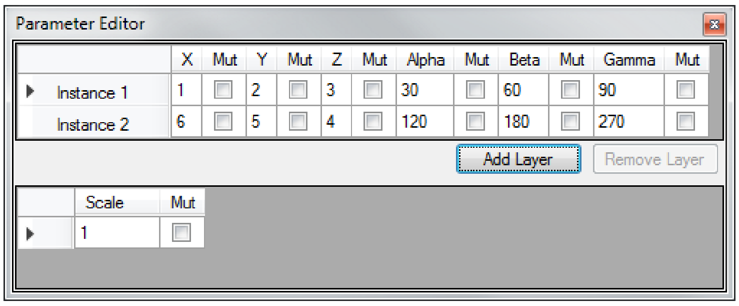
	Scale	Mut
	1	☐


Error

ERROR while computing: The current integration method was not implemented for CPU usage (probably means you selected VEGAS and a space-filling symmetry, which haven't been implemented on a GPU).. Details: The current integration method was not implemented for CPU usage (probably means you selected VEGAS and a space-filling symmetry, which haven't been implemented on a GPU).

`Space Filing Symmetry` does not automatically center the structure around the origin. To facilitate easer calculations, it is suggested to wrap the entire structure in a `Manual symmetry`, and place the center of mass of the entire structure approximately at the origin. Note: An error message will appear when trying to use `Grids` for `Space Filling Symmetry` (which was implemented only for CPU) of a PDB file, and compute orientation average using VEGAS Monte Carlo (implemented only for GPU). This computation, however, can be done using the Hybrid or Direct methods. If geometric models are used (instead of a PDB), the computation can be done.

4.3 Manual Symmetry


Parameter Editor

	X	Mut	Y	Mut	Z	Mut	Alpha	Mut	Beta	Mut	Gamma	Mut
Instance 1	1	☐	2	☐	3	☐	30	☐	60	☐	90	☐
Instance 2	6	☐	5	☐	4	☐	120	☐	180	☐	270	☐

	Scale	Mut
	1	☐

Any three dimensional object in 3D space can be positioned and rotated by a set of six parameters: $\{x, y, z\}$ to determine its displacement and $\{\alpha, \beta, \gamma\}$ as Euler angles for rotation. A `Manual Symmetry` is a table of these six parameters per object. In the `Parameter Editor` Figure shown, there are two copies of children (`Instance 1` and `Instance 2`) at two arbitrary positions and orientations. The list of coordinates and orientations can be loaded from a "Docking List" (DOL) text file (`*.dol`) only after adding a `Manual Symmetry`. The file should be tab or space delimited with each row starting with an index, followed by $\{x, y, z\}$ and finishing with $\{\alpha, \beta, \gamma\}$. A file that would create the `Manual Symmetry` in the image would look like:

```
1    1    2    3    30     60     90
2    6    5    4    120    180    270
```

Note: The indices at the beginning of each line have no meaning, they just need to be integers.

4.4 Scripted Symmetry

A scripted symmetry is a `Lua` script that creates a Manual Symmetry behind the scenes based on parameters provided by the user. The `Lua` script has to contain an `Information` Table and several functions that will be presented below. We shall run through an example script to show the usage.

`Information` Table

The `Information` Table has five fields: `Name`; `Type`; number of layer parameters `(NLP)` ; `MinLayers`; `MaxLayers`.

```
Information = {
    Name = "Simple Grid in XY", -- This is the name that will be displayed in the
        ↪ Domain View
    Type = "Symmetry", -- This is the type, should be "Symmetry" for scripted symmetries
    NLP = 2, -- Number of Layer Parameters: The number of parameters per layer
    MinLayers = 2, -- The minimal number of layers (<= MaxLayers)
    MaxLayers = 2, -- The maximal number of layers (>= MinLayers)
};
```

The parameters provided by the user are organized in layers, in each layer the same number of parameters `(NLP)`. (defining `MaxLayers` to be −1 enables the user to control the number of layers from the GUI)

`Populate` Function

Next, several functions need to be added. The first function to implement is the workhouse of the script. It populates the `Manual Symmetry`.

```
function Populate(p, nlayers)
    -- This is really just a sanity check, but doesn't hurt to add it.
    if (p == nil or nlayers ~= 2 or table.getn(p[1]) ~= 2) then
        error("Parameter matrix must be 7x1, it is " .. nlayers .. "x" ..
            ↪ table.getn(p[1]));
    end
```

This is a check for the correct size of the parameter matrix, the number of layers (nlayers) and the number of parameters in each layer (table.getn(p[1]))

```
    -- Create meaningful names
    DistanceX     = p[1][1];
    DistanceY     = p[2][1];
    RepetitionsX = p[1][2];
    RepetitionsY = p[2][2];


    -- Create the table to return, and the indices
    res = {};
    n = 1;


    -- Fill the res table to papulate Manual Symmetry

    for y = 1, RepetitionsY  do
            for x = 1, RepetitionsX  do
                res[n] = {DistanceX*(x-RepetitionsX/2-0.5),
                          DistanceY*(y-RepetitionsY/2-0.5),
                           0,0,0,0};

                    n = n + 1;
            end
    end

    -- Return the complete table of x,y,z, \alpha, \beta, \gamma
    return res;

end
```

notice that to reduce the `Grid Size` required for accurate calculations, we were careful to center the complete object around the origin.

`GetLayerName` Function

There are a few additional functions that are required for the user interface (UI). It needs to know things about variables like `Layer` and `Parameter Names`, is it applicable, and default values. For starters, the text displayed to the left of the table in Figure 2.2 (section 2.6) is generated with the function below. Each row (or layer) gets the text based on its index, as shown.

```
function GetLayerName(index)
    if index == 0 then
        return "X";
    elseif index == 1 then
        return "Y";

    else
        return "N/A";
    end
end
```

`GetLayerParameterName` Function

Next, the text above the table in Figure 2.2 is obtained with the function below. Each column (or layer parameter) gets the text based on its index, as shown.

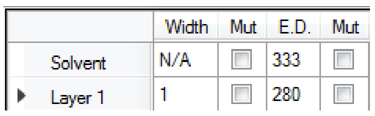
```
function GetLayerParameterName(index)
    if index == 0 then
        return ”distance”;
        elseif index == 1 then
        return ”Repetitions”;
    else
        return ”N/A”
    end
end
```

`IsParamApplicable` Function

The applicability of a parameter is less obvious in this example. Here, all the parameters are applicable, so the function is a straightforward `return true;`.

```
function IsParamApplicable(layer, layerParam)
    return true;
end
```

A better (theoretical) example would be as shown in the image here. In this example, the `Solvent` layer does not have a `Width`. So the function should return true for all options *except* when both `layer` and `layerParam` equal zero.



	Width	Mut	E.D.	Mut
Solvent	N/A	☐	333	☐
▸ Layer 1	1	☐	280	☐

```
function IsParamApplicable(layer, layerParam)
    if (layer == 0 and layerParam == 0) then
        return false;
    return true;
end
```

`GetDefaultValue` Function

Finally, the UI needs to have some default values for each of the `layer - layerParam` combinations.

```
function GetDefaultValue(layer, layerParam)
    if layer == 0 then
        if layerParam == 0 then
```

```
            return 1;
        elseif layerParam == 1 then
            return 10;
        end
    elseif layer == 1 then
        if layerParam == 0 then
            return 1;
        elseif layerParam == 1 then
            return 10;
        end

    end
end
```

Chapter 5

Lua Scripts

It's nice to finally get scripts offered to me that aren't the ones Tom Hanks wipes his butt with.

Jim Carrey

Scripting in D+ is done with the Lua language. Scripts can be used to create Symmetries (covered in chapter 4) and to do repetitious work. It can also be used as a general `Lua` environment. There are several examples available in the `./LuaScripts/` directory, where D+ is installed. This chapter will address the bound D+ functions that have been exposed for use.

Most of the functions are gathered in a table called `dplus`. The list of Dplus Lua commands are provided in the next section. Additional explanations and usage examples are provided in the two sections that follow.

5.1 Complete List of Dplus Lua Commands

The list of D+ `Lua` commands can be seen in the `Command Window` of D+ by typing `dplus.`, and holding the `Ctrl` and `Space` keys.

In the following we present the complete list of D+ `Lua` commands and their arguments or parameters. Optional parameters are marked with [brackets].

```
dplus.openconsole()
```

Opens an external console window (for performance and debugging purposes)

```
dplus.closeconsole()
```

Closes the external console window

```
dplus.getparametertree()
```

Returns the current parameters as a parameter tree object

```
dplus.setparametertree(param_tree)
```

Sets the parameter tree as the current parameters

```
dplus.save(filename)
```

Saves parameters to .state file

```
dplus.load(filename)
```

Loads parameters from .state file

```
dplus.findmodel(name[, container])
```

Finds a (geometric) model by name, optionally with an external container (DLL) filename

```
dplus.findamp(name[, container])
```

Finds an amplitude model by name, optionally with an external container (DLL) filename

```
dplus.generate([param_tree, save_filename, should_save_amplitudes])
```

Generate using the given parameter tree (or nil for current parameters), optionally saving the result to a file. 'should_save_amplitudes' determines whether D+ should save the grid amplitudes while generating.

```
dplus.fit(data, param_tree, properties)
```

Fit a given parameter tree (or nil for current parameters) to the given data table, using the fitting properties given in 'properties'.

```
dplus.readdata(filename)
```

Reads data from a signal file to a Lua table

```
dplus.writedata(filename, data)
```

Writes data from a Lua table to a signal file (TSV format)

```
dplus.figure(title, xlabel, ylabel)
```

Opens a new figure with a given title, x, and y labels. Returns figure ID for use in dplus.showgraph*

```
dplus.showgraph(data, figure_id, color)
```

Plots a signal filename (if data is a string) or table (if data is a Lua table) on a given figure ID, using a specific color (given as a string of the color name). If 'figure_id' is nil, uses the last opened figure.

```
dplus.{msgbox,mbox,message,messagebox}(message)
```

Opens a message box (good for user notifications)

```
dplus.{sleep, wait}(time_ms)
```

Wait for the given amount of milliseconds

5.2 UI Related Functions

Some of the functions allow some basic control of the user interface. Of these, some have more complex uses and will be addressed in separate sections.

Opening and closing the console The console can be opened and closed via the commands `dplus.openconsole()` and `dplus.closeconsole()` respectively.

Saving and loading parameter files The entire set of model, fitting and camera parameters can be saved to a file[1]. `dplus.save(string)` saves the current set of parameters to a file. For example,

```
dplus.save("C:\\path\\to\\file.state")
```

will save the current state to `C:\path\to\file.state`. Likewise,

```
dplus.load("C:\\path\\to\\file.state")
```

loads the file into the UI and replaces any existing parameters.

Reading scattering curve data files Loading a signal to the `2D graph` is achieved by

```
dplus.readdata(filename)
```

Message boxes A (blocking) dialog box with a specific message can be popped up with any of the following (without `dplus.`): `msgbox(string)`, `mbox(string)`, `message(string)`, or `messagebox(string)`.

Navigating Lua tables Owing to the fact that `Lua` stores *everything* in a Key-Value table, it is often necessary to find out what the keys are. Using the provided `PrintKeys(table)` prints a list of the table keys in the `Command Window`.

Waiting Two identical functions for waiting are provided: `sleep(ms)` and `wait(ms)`. Both accept an integer and pause the script for that number of milliseconds.

5.3 Scripting Related Functions

When running repetitive tasks (such as generating all the permutations of four parameters at increments of X) several types of functions are needed, such as manipulating the parameters, generating models, saving results, etc.

Obtaining the parameters from the GUI While it is possible to create the parameters from scratch using a script, a better method would be to obtain the parameters from the GUI, using `dplus.getparametertree()`. The `PrintKeys` function can be used to navigate and to obtained tree like so (in the Command Window):

```
myTree = dplus.getparametertree()
dummyVar = PrintKeys(myTree)
--[[ Prints:
```

[1]The saved file is itself a `Lua` script that contains the parameters in tables.

```
    Size: 4
    Scale
    ScaleMut
    Geometry
    Populations
]]
dummyVar = PrintKeys(myTree.Populations)
--[[ Prints:
    Size: 1
    1
]]
dummyVar = PrintKeys(myTree.Populations[1])
--[[ Prints:
    Size: 3
    PopulationSizeMut
    PopulationSize
    Models
]]
-- etc.
```

Setting the parameters to the GUI After navigating the tree and changing one of the parameters like

```
myTree.Populations[1].Models[1].Parameters[2][2] = math.pi^math.exp(1)
```

the GUI remains ignorant of the change until it's updated. One way of updating the GUI is to use `dplus.setparametertree(myTree)`. NOTE: This clears all existing entities/models and creates new ones. If one of the models was a PDB model, then the PDB file will be read again.

Updating the GUI's parameters If all that needs to be done is change a few parameters without *adding or removing* parameters, the better option would be to update the GUI parameters. This way, the aforementioned PDB model does not get reread and any information D+ has created for the model is not lost. This is done with `dplus.updateparametertree(myTree)`.

Generating models Modifying parameters is all good, but using them to generate model signals is better. The command is `dplus.generate` and has three different overloads (usage forms). All forms return a table which contains the (x, y) coordinates of the resulting scattering curve graph. The simple form is `res = dplus.generate()`. This takes the parameters from the GUI and uses them to generate a model of a scattering curve signal. It is also possible to generate a model without updating the GUI via `res = dplus.generate(myTree, saveFileName)`. There are some times when one might want to save the amplitude files of the generated model. This can be achieved by `res = dplus.generate(myTree, saveFileName, true)`.

Saving files Generating 10^{35} models is better, but wouldn't it be amazing if the results were save to a file? Again, achieved like so (including a bit of error checking):

```
res = dplus.generate()
ddd = dplus.writedata(saveFileName, res)

if not ddd then
    print("There was a problem with file " .. saveFileName .. ".\n");
end
```

Fitting If saving 10^{35} files is amazing, fitting and selecting one file to save would be awesome. A Lua interface for this purpose exists and works. The challenge is to choose the right starting fitting parameters. D+ adopted `Ceres-Solver` fitting algorithm using Riddler's method for derivation.

Miscellaneous

In Lua assigning a table to a variable does not copy it. If we did `a = myTree` and then changed a value either in `a` or in `myTree` the change will be apparent in the other variable. An actual copy function is provided and used as `a = table.copy(myTree)`.

Chapter 6

Integration Methods

Art is an attempt to integrate evil.

Simone de Beauvoir

The generation of a model signal in D+ requires the calculation of

$$I(q) = \frac{1}{4\pi} \int_{\Omega_q} |F(\vec{q})|^2 \, d\Omega_q \tag{6.1}$$

where the $\int_{\Omega_q} d\Omega_q$ integration (orientation average in reciprocal q-space) is done numerically. The solid-angle in reciprocal q-space, Ω_q, is essentially two dimensions (containing the polar-angle $\theta_q = [0, \pi]$ and the azimuth-angle $\phi_q = [0, 2\pi]$), defining the orientations.

There are two options in the `Preferences` pane (see sec. 2.10) that affect the results of the quadrature. The main items are `Integration Iterations` and `Convergence`. The `Integration Iterations` parameter is the maximal number of evaluations (the default value is `10,000,000`), unless otherwise is specified in the following sections. The `Convergence` parameter, ϵ, is a value that allows the evaluation to be stopped once converged.

There are many quadrature (numerical integration) methods, each with its own set of advantages and disadvantages. D+ has a few methods to choose from.

6.1 Monte Carlo

The default methods is `Monte Carlo (Mersenne Twister)` integration. It is implemented both for a CPU and a GPU, although there are slight differences in the precise implementations. In this case the intensity is given by

$$I(q) = \frac{1}{w} \sum_{i}^{w} |F(\vec{q}_i)|^2 \tag{6.2}$$

where $\vec{q}_i = \left(q, \theta_q^i, \phi_q^i\right)$ and the values for θ_q^i and ϕ_q^i are chosen randomly.

The convergence parameter is

$$\left|1 - \frac{I_w}{I_{w-k}}\right| < \epsilon \tag{6.3}$$

w and k are integers and $1 \leq k < w$. In practice, the intensity is compared to multiple previous points to determine convergence.

Technical note: The random number generator used is Mersenne Twister (mt19937).

6.2 Adaptive (VEGAS) Monte Carlo

The basic idea of `Adaptive (VEGAS) Monte Carlo` integration is to divide the integration axes into sections such that the variance in the variance is minimized. This can speed up the convergence of the quadrature if the variance changes in parallel to one of the integration axes. If the structure is not parallel to either of the axes, then the speed-up will be less, and can be even slower than the classic Monte Carlo method.

The convergence parameter is

$$\left|1 - \frac{I_w}{I_{w-k}}\right| < \epsilon \tag{6.4}$$

Here too the intensity is compared with multiple previous points to determine convergence.

This method is implemented for the GPU only at this point. VEGAS is particularly useful for computing the models of structures that have one dimension with significantly different length then the other dimensions (a rod-like or tubular structure, for example). The scattering amplitudes of these objects will be concentrated around one or two axes in reciprocal space, hence will benefit from the adaptive integration approach. When using `VEGAS`, the scattering `Amplitude` cannot be exported (see sec. 8.2).

6.3 Adaptive Gauss Kronrod

`Adaptive Gauss Kronrod` is a recursive method that subdivides the integration region as long as the current region has an estimated error greater than ϵ. In this method, the `Integration Iterations` is not the number of evaluations, but rather the maximum recursion depth. Note that in contrast to the number of iterations which is very large (the default is `10,000,000`), the recursion depth should be `10` or `15`. **Make sure to change this value!**.

The `Adaptive Gauss Kronrod` method used is a 7-point Gauss rule with a 15-point Kronrod rule Kronrod 1964; Laurie 1997. The error estimate is given by the relative difference between the two:

$$\left|1 - \frac{G_7}{K_{15}}\right| < \epsilon \tag{6.5}$$

This method is deterministic and was implemented only for the CPU. It should be used for structures that have one dimension with significantly different length then the other dimensions. When using `Adaptive Gauss Krorrod`, usually the `Convergence` value can be ten times higher than the value in `Monte Carlo (Mersenne Twister)` for similar results. It is often enough to compute fewer q points (about 10 or 20%) to get faster results.

Chapter 7

Reciprocal Grids and the Hybrid Method

Love is a reciprocal torture.

Marcel Proust

7.1 Background

D+ attains its speed by first calculating the scattering amplitudes at the relevant 3D reciprocal $\vec{q}$-space of repeating subunit objects, saving them in memory, and then summing up the contribution from their multiple copies (provided by their symmetries). Using saved repeated subunit objects in memory saves the need to calculate the contribution to the scattering amplitude from each of the repeating subunits, replacing many of the calculations with lookups at the precalculated reciprocal grid (*i.e.* lookup table) of the repeating subunit plus interpolation for $\vec{q}$-values between the precalculated points. As with all lookup tables, the density has to be high enough to sample all the variations and features of the scattering amplitudes.

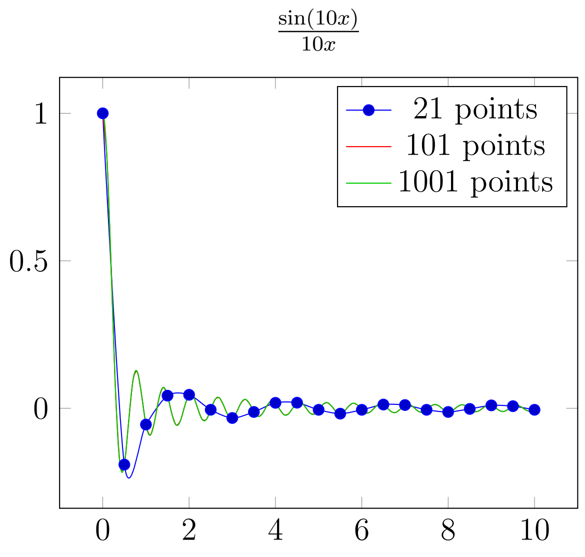


Figure 7.1: Plots of an oscillatory function at different sampling rates.

If the largest distance within the object is L, the scattering amplitude will oscillate approximately like $\sin\left(q \cdot \frac{L}{2}\right)$. The density of the points in the reciprocal-space lookup table (or grid) must be greater than two points per oscillation. This is known as the NyquistShannon sampling rate. Hence, the grid density should be at least $(\Delta q)^3 \approx (2\pi/L)^3$ and the 3D reciprocal-space grid-size should be at least $G \approx \left(q_{\max} \cdot \frac{L}{2\pi}\right)^3$, where $q_{\max}$ is the maximum q-range.

Alongside the installation of D+, there is a small application dubbed `Suggest Parameters` that can be opened from the main window under the `Settings` menu. This application can be used to give an estimate of the necessary grid size based on the x, y, z, and q_{max} parameters.

As an example, consider the plots in Figure 7.1. The blue, red and green curves all calculated $\frac{\sin(10x)}{10x}$. The difference is only in how many points were sampled. In the case of the blue curve, there is an under-sampling that results in inaccurate values in the regions requiring interpolation (between the blue dots). In order to accurately represent the sinc function, the sampling density (or in our phrasing, grid density) must be increased. The flip side is that there is little accuracy gained by increasing the sampling to 1001 points instead of 101. Whereas the negligibly increased accuracy itself does not hurt, it costs both computation time and memory.

The Fourier space (scattering amplitude) of large repetitive objects becomes increasingly oscillatory with the size/repetitions. Therefore, creating a reciprocal grid (from here on, RG) of the large object would require a very dense grid. This makes naïvely using an RG for large objects dangerous as the interpolated values will have little to do with the underlying function, just like the blue curve in Figure 7.1.

If the large/long object is represented by a single PDB file or a single geometric model, there is (currently) little to do. If, however, the object is constructed of smaller repeating units, which *can* be represented by an RG, there is a solution.

Hybrid Calculations

For the Hybrid method to work, the structure must be described in a hierarchical manner. For example, consider a series of blue spheres lined up in a linear fashion, as shown in Figure 7.2. There are 12 uniform spheres of the same radius spaced $2r = 2\,\text{nm}$ apart. Using an RG with a linear density of $0.1875\,\text{nm}^{-1}$ (81 points between $-7.5\,\text{nm}^{-1}$ and $7.5\,\text{nm}^{-1}$) the scattering of single sphere can be calculated accurately. If, however, the same RG is used to compute the scattering of all 12 spheres, then the Nyquist criteria is not met. The resulting undersampling causes little oscillations in the generated signal with a period of the RG density, as seen in the blue curve in Figure 7.2.

There are three ways to deal with this. The simplest and *wrong* way would be to increase the RG density. The minimal density needed is $G \approx \left(q_{\max} \cdot \frac{L}{2\pi}\right)^3$ where L is the largest distance within the modeled structure. For a single sphere, L is $2r$, whereas for the ensemble of twelve spheres L is $24r$. The calculation time and memory requirements scale like L^3 and therefore this method is not recommended for twelve spheres.

The second way is to use the Direct method. In the Direct method, each type of geometry or PDB object is identified, and all its copies in the structure as well as their orientations and

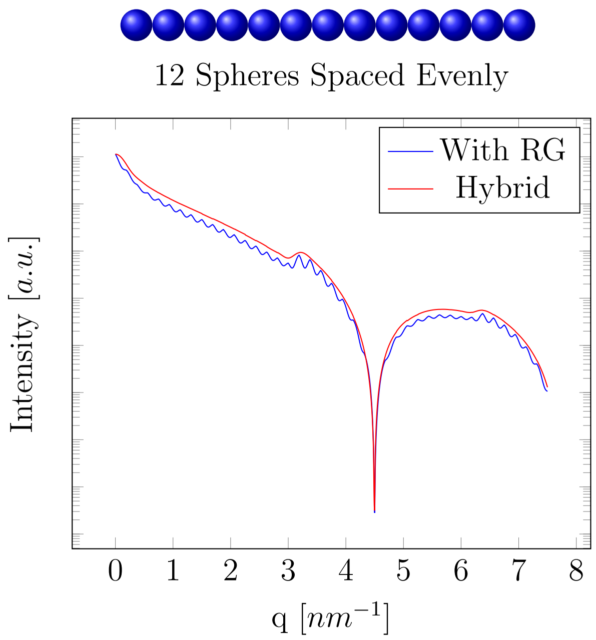


Figure 7.2: Comparing the RG with the Hybrid method for the case of 12 spheres that are evenly spaced along a line.

locations are collected. The intensity is then directly computed by modifying Equation 4.1 so that each node in the tree structure is grouped by its orientation. If the same orientation (j, m) is observed in repeating subunits in the structure, the contribution of that subunit to the total scattering amplitude is reused, with the relevant phase contribution associated with its real-space locations, given by the vectors $\vec{R}_{j,m,k}$:

$$F(\vec{q}) = \sum_{j=1}^{J} \sum_{m=1}^{M_j^u} \left[F_j \left(\mathbf{A}_{j,m}^{-1} \vec{q} \right) \cdot \sum_{k=1}^{K_{j,m}} \exp \left(i\vec{q} \cdot \vec{R}_{j,m,k} \right) \right] \tag{7.1}$$

J is the number of different types of objects (leaves in the hierarchical tree, see Figure 4.1), M_j^u is the number of unique orientations of object type j, determined by the rotation matrices $\mathbf{A}_{j,m}$. $K_{j,m}$ is the number of real-space translations of object j in orientation $\mathbf{A}_{j,m}$. Orientation averaging is then computed by Equation 6.2. In D+, the Direct method was combined with the RG method. This combination is called the Hybrid method and it is the best solution for the case of large structures. Currently, the Direct method is implemented only for the CPU.

In the Hybrid method, RGs are computed from the leaves up to predetermined nodes in the assembly hierarchical data tree structure (Figure 4.1), and from these nodes upward computed as subunits in the Direct method.

Differently oriented subunits are directly computed as in the Direct method. The highest calculated RGs, however, are accessed as $F(\mathbf{A}^{-1}\vec{q})$, as in the RG method. In the Hybrid method, at least one of the leaves in the hierarchical data tree structure (a geometric or a PDB model) is calculated to an RG. RGs may be computed for internal nodes and if they are then their leaves are discarded and the internal nodes are treated as leaves.

The Hybrid method averts the use of RGs that represent long structures, thus providing a solution with arbitrary accuracy in the face of the RG memory/accuracy trade-off. This approach is effectively a coarse-grained method with near-atomic resolution accuracy, where the subunit count (n) can be much smaller than the number of atoms. A rigorous analysis of the method is published in Ginsburg et al. 2016. The advantage of the Hybrid method is that the grid size depends on the length of the largest subunit, for which grids are calculated, rather than the length of the entire structure. Going back to our example, the preferred method is the Hybrid method in which we compute an RG for a single subunit (sphere) and use it for computing the scattering of the ensemble without an intermediate RG. More details about the algorithm can be found in Ginsburg et al. 2016; Ginsburg et al. 2018.

7.2 Using the Hybrid Method in D+

In practice, you can tell D+ to use the Hybrid method from a given point in a hierarchy by deselecting the `Use Grid From Here` checkbox in the `Symmetry Editor` pane from the node that you do not want to use an RG. For example, assume you wanted to model the 12 aforementioned spheres using an RG for the sphere and not for the linear spacing. You would click on the `Sphere` in the `Domain View` pane, ensuring that the `Use Grid From Here` checkbox in the `Symmetry Editor` pane was checked. You would then click on the symmetry you used to create the linear spacing in the `Domain View` and uncheck the `Use Grid From Here` checkbox in the `Symmetry Editor` pane. After pressing generate a signal similar to the red curve in Figure 7.2 will be computed.

Chapter 8

Examples

Few things are harder to put up with than the annoyance of a good example.

Mark Twain

D+ is very dynamic. Demonstrating all the options combinations would be near impossible. Examples of how to use scripts can be found in the `./LuaScripts/` directory. Other examples, which were discussed in the paper and its SOM can be found in the `./Example Files/` directory, where D+ is installed. The folder contains 7 subfolders. Each folder, contains a `*.state` file that can be loaded to D+. After the `state` file is loaded, pressing `Generate` will compute the scattering curve. The expected result is also provided as an `*.out` file. Here, we shall present two examples of models: one real and PDB based; and one for fun and to demonstrate the great flexibility of D+.

8.1 Microtubles

The original reason D+ was developed was to model microtubules at resolutions that matched the SAXS data. Microtubles are comprised of $\alpha\beta$ tubulin heterodimers arranged as a tube on a helical lattice. There are many PDBs of the tubulin dimer. We shall use a modified PDB file of the tubulin dimer located in the `./Example Files/4_Microtubule` directory: `3j6f_Dimer_2_2_Added_H_GCentered.pdb`. The PDB file is aligned such that the long axis is roughly parallel to the z axis (details about the alignment of atomic structures can be found in the supporting online materials (SOM) of Ginsburg et al. 2018).

Load the PDB (in the `Domain View`). The PDB is already centered, so checking or unchecking the Center PDB checkbox will have no effect. To make the multiple copies of tubulin that are needed to construct a microtubule, add the `Super Helical Left Hand MT Helix.lua` scripted symmetry located in the `./LuaScripts/` directory. In the `Domain View`, drag the PDB line onto the `Super Helical Left Hand MT Helix` line. This tells D+ to use the script to make multiple copies of the tubulin dimer.

To better understand how the `Scripted Symmetry` works, play around with the parameters. Then look at the code. You can see the code of a loaded `Scripted Symmetry` in the `Script Editor` by double clicking on the script name in the `Domain View`.

Now, let us correct the model. Select the `Super Helical Left Hand MT Helix` in the `Domain View`. Enter the parameters from the table in the `Parameter Editor`. These are roughly the correct parameters for microtubules. Pressing `Generate` (in the Controls pane) may or may not give an accurate scattering pattern (depending on parameters like `Grid Size`, `Convergence`, and `Integration Iterations`), as the object is long and artifacts will occur (see Chapter 7 for details).

8.2 Interim Results

Assume that in the previous section, you did not choose to integrate using VEGAS nor opted to use the Hybrid Method. These choices are important for reasons that will be explained below. There are two results besides the actual scattering curves once the generate button has been pressed. First, as we constructed a larger atomic model from a PDB file and a symmetry, one might want a PDB file of the entire structure. This PDB file can be obtained by selecting the symmetry whose PDB file you want and then under the `File` menu choosing `Export PDB Representation`. The PDB file is constructed using the parameters when the generate button is pressed, so if you change the parameters after pressing generate, the resulting file may not be what you would expect.

Parameter	**Value**
Radius	11.94
Pitch	12.195
Units per Pitch	14
Units in Pitch	14
Discrete Height	36
# Helix Starts	3
Super Helical Pitch	0

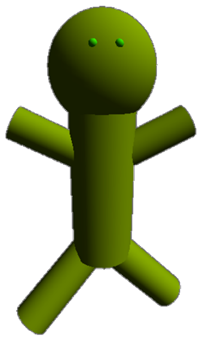
The second result that can save computation time later, is the scattering amplitude of any of the elements in the tree (in this case either the PDB model or the `Scripted Symmetry`). Amplitude can be saved under the `File` menu, by pressing on `Export Amplitude File...`. If the tree has more than one model, pressing on each model before exporting the amplitude, will save the amplitude of the chosen model. If models are under `Manual Symmetry`, pressing on the `Manual Symmetry` or on any of the models in it, before exporting the amplitude, will save the amplitude grid of the entire symmetry.

A saved amplitude (RG) can be loaded as an `Amplitude Grid (AMP)` in the `Domain View` pane. This `Amplitude Grid` saves the computation time of that (sub)structure in the future as the interim result can be loaded and inserted in other structures as well.

When using the `Adaptive (VEGAS) Monte Carlo` integration method, all the calculations are carried out on the GPU without transferring the interim results back to the CPU. Once the computation is complete, the interim results are lost. In particular, the `Amplitude Grid` file (an `AMP` file) cannot be exported. Similarly, when using the `Hybrid Method`, no RG is created for nodes whose `Use Grid From Here` checkbox is unchecked and therefore it cannot be saved.

8.3 Stick Figure

Make a stick figure. For the head, add a sphere, for the body, a cylinder, etc. Use this picture as a guide. Try to use both geometric shapes and symmetries (manual and space-filling). The method we used can be found in `./Example Files/D+Man.state`. There is no scientific value to this structure, it is just a way to practice manipulating objects in D+.



Chapter 9

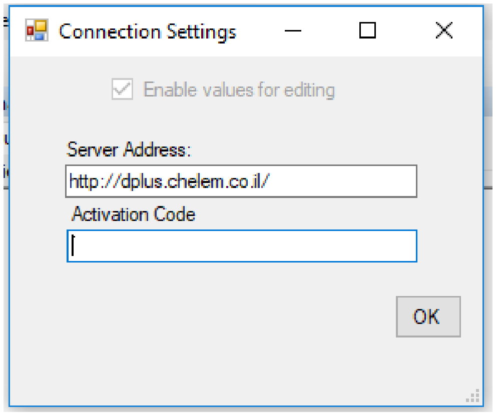
Remote Calculations

Magnetism is one of the Six Fundamental Forces of the Universe, with the other five being Gravity, Duct Tape, Whining, Remote Control, and The Force That Pulls Dogs Toward The Groins Of Strangers.

Dave Barry

In the `Start Menu`, under `D+`, there is a `Remote D+` option. If you do not find this option, go to `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\D+`, and run `Remote D+` from there.

The main difference between `Remote D+` and the local version of D+ is where the computations take place. In the local version, a local copy of the backend is run, using your computers resources. In the remote version, computations are carried out on a server running the backend with the communications handled over the internet.



Server Address: http://dplus.chelem.co.il/
Activation Code

In the directory with `DPlus.exe` (or `%appdata%`) there may be an additional file named `Server.dserv`. This is a plain text file with the servers URL and an optional port number. If this file does not exist, then a window will open when starting D+ that looks like the one to the right, just without the Server Address. In the `Server Address` field, enter the servers URL (and port number if it is different than 80).

When subsequently starting D+, it will attempt to connect to the same server. In order to change the server, either delete the `Server.dserv` file before launching, or choose `Configure Server` under the `Settings` menu.

You will then need an activation code like this: `fd8d997a007d96a320d556846a48d340315faf2e`. If you do not have a GPU on your computer, running long calculations will be faster on a server with GPU.

Bibliography

Agarwal, Sameer, Keir Mierle, and Others. *Ceres Solver*. `http://ceres-solver.org`.

Als-Nielsen, Jens and Des McMorrow (2011). *Elements of modern x-ray physics*. John Wiley & Sons.

Ben-Nun, Tal et al. (2010). "*X+*: a comprehensive computationally accelerated structure analysis tool for solution x-ray scattering from supramolecular self-assemblies". In: *Journal of Applied Crystallography* 43.6, pp. 1522–1531. DOI: `10.1107/S0021889810032772`.

Chen, Vincent B. et al. (2010). "*MolProbity*: all-atom structure validation for macromolecular crystallography". In: *Acta Crystallographica Section D* 66.1, pp. 12–21. DOI: `10.1107/S0907444909042073`.

Fraser, R.D.B., T.P. MacRae, and E. Suzuki (1978). "An improved method for calculating the contribution of solvent to the X-ray diffraction pattern of biological molecules". In: *Journal of Applied Crystallography* 11.6, pp. 693–694.

Ginsburg, Avi et al. (2016). "Reciprocal Grids: A Hierarchical Algorithm for Computing Solution X-ray Scattering Curves from Supramolecular Complexes at High Resolution". In: *Journal of Chemical Information and Modeling* 56.8, pp. 1518–1527. DOI: `10.1021/acs.jcim.6b00159`.

Ginsburg, Avi et al. (2018). "D+: Software for High-Resolution Hierarchical Modeling of Solution X-Ray Scattering from Complex Structures". In: *Submitted* 0.0, pp. 0–0.

Hamilton, WC (1974). "International Tables for X-ray Crystallography, vol. IV". In: *Birmingham: Kynoch Press.(Present distributor Kluwer Academic Publishers, Dordrecht. Table 2.2B.)* Pp. 273–284.

Kittel, C. (2005). *Introducion to Solid State Physics.* John Wiley and Sons.

Kronrod, AS (1964). "Integration With Accuracy Control". In: *DOKLADY AKADEMII NAUK SSSR* 154.2, p. 283.

Laurie, Dirk (1997). "Calculation of Gauss-Kronrod quadrature rules". In: *Mathematics of Computation of the American Mathematical Society* 66.219, pp. 1133–1145.

Marsh, RE and KL Slagle (1983). "Corrections to table 2.2 B of volume IV of International Tables for X-ray Crystallography". In: *Acta Crystallographica Section A: Foundations of Crystallography* 39.1, pp. 173–173.

Pavlov, M Yu and BA Fedorov (1983). "Improved technique for calculating X-ray scattering intensity of biopolymers in solution: Evaluation of the form, volume, and surface of a particle". In: *Biopolymers* 22.6, pp. 1507–1522.

Schneidman-Duhovny, Dina et al. (2013). “Accurate SAXS profile computation and its assessment by contrast variation experiments”. In: *Biophysical journal* 105.4, pp. 962–974.

Schrödinger, LLC (2015). “The PyMOL Molecular Graphics System, Version 1.8”.

Svergun, D, Claudio Barberato, and Michel HJ Koch (1995). “CRYSOL – a program to evaluate X-ray solution scattering of biological macromolecules from atomic coordinates”. In: *Journal of Applied Crystallography* 28.6, pp. 768–773.

Székely, Pablo et al. (2010). “Solution x-ray scattering form factors of supramolecular self-assembled structures”. In: *Langmuir* 26.16, pp. 13110–13129.