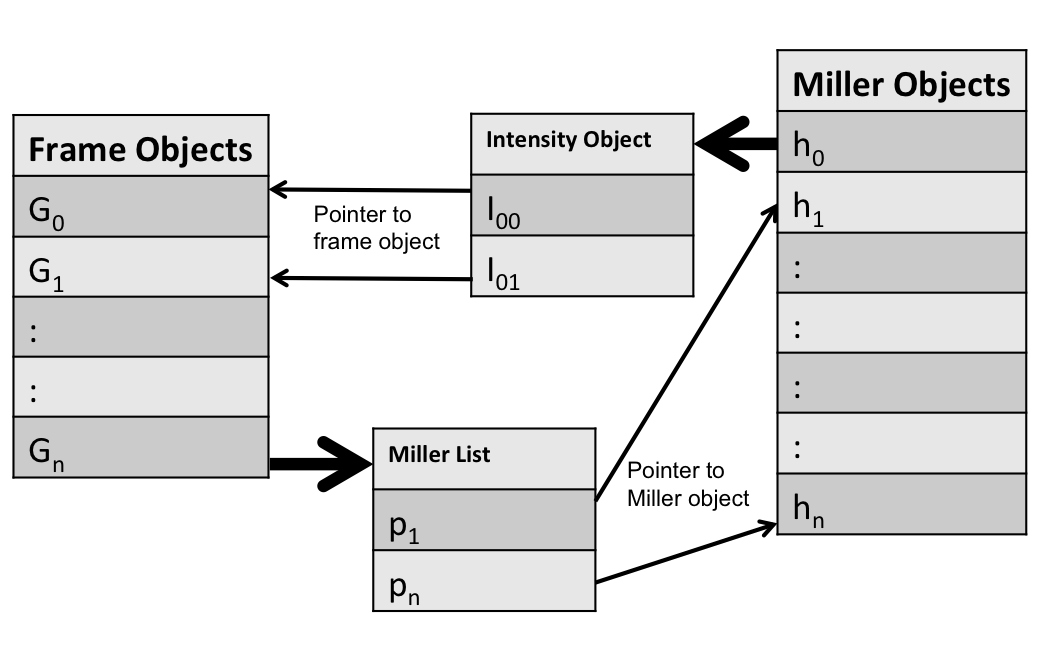
**Scaling for serial femtosecond crystallographic**

In serial femtosecond crystallographic experiment, each diffraction image is collected from different crystal. In addition, the incident beam intensity varies for each shot. Thus, scaling is required to put intensities from different frames on the same scale. The target function to be minimized (Hamilton et al., 1965):



The *Ih* and its derivatives need to be evaluated for each term in the summation at each optimization step to calculate the function and its derivatives with respect to given Gm, which is computationally expensive where *n* is the number of observations and *m* is the number of frames. However, if a given miller index is not observed in a given frame, then the term is zero for this Gm’*.*



In the data structure shown in the figure above each miller object has a list of observed intensities where each intensity object has a pointer attribute pointing to the frame object contains the observed intensity. In addition, each frame object has a list of pointers to the miller objects with their index observed in this frame. Thus, when evaluating the derivatives , we don’t need to calculated the derivatives of all terms with respect to Gm’ but only the terms *Ih*(G*m’*) that has observations in the in frame m’ need to be evaluated.

Another approach is to treat the Gm’’s and *Ih’s* as independent variables and refine all of them together. If Gm’’s and *Ih’s* are independent variables, the complexity of the algorithm needed to evaluate the derivatives is linear in the number of observations n .



However, because there are many thousands of frames in the standard XFEL experiment, the number of parameters to be refined is huge especially for crystals with large unit cell and requires many optimization steps before convergence.

The Levenberg-Marquardt algorithm introduces a damping factor that is adjusted through the optimization, which accelerate the convergence. If there is rapid decrease in the residuals, the value of the damping factor is small and the algorithm is converging to the Gauss-Newton method. If the reduction in the residuals is slow then the value of the damping factor is increased and the algorithm is converging to gradient descent.

For each iteration in the optimization algorithm, the normal equation is solved by matrix decomposition, forward and backs substitution. However, the decomposition of the normal matrix is computationally expensive , where *n* is the number of parameters. In addition, constructing the normal matrix requires calculating , where J is the Jacobian matrix. The complexity of this calculation is , where m is the number of observations and n is the number of parameters. For matrix J, where *Ih* and *Gm* are independent each row has only two nonzero element (the term only dependent on the parameters G0 and I1). Thus, if only the nonzero elements are considered the complexity of the matrix multiplication is , where is the number of the non zero elements in each row.

The A matrix is decomposed into a product of two matrices , where *L* is a lower triangular matrix with real and positive diagonal elements. The elements of *L* are evaluated according to the following equations:

The number of operations required to calculate *L* for dense matrix is [1]:

For the sparse matrix the complexity depends on the number of nonzero elements in the L matrix and the number of operations required [1]:

Thus, to efficiently calculate *L*, the number of the nonzero elements in *L* has to be determinant initially. However, when the normal matrix A is decomposed into the lower triangular matrix *L* the fill-in problem arises. That means there are some zero elements in A that are nonzero in *L* and the nonzero elements of *L* has to be known prior to evaluating equations 7 and 8.

To identify the nonzero elements of a sparse *L* matrix the *graph elimination algorithms* maybe used [2]. Briefly, the G(A) is a undirected graph of the symmetric A matrix with n vertices having edges between the nonzero elements only. For example, there is an edge between vertex i and j only if aij in the A matrix is nonzero. To identify the fills (the elements that were initially zero in A but they are nonzero in *L*), a vertex is eliminated from the graph then the neighbors of the eliminated vertex become fully connected graph (clique). These new connections define the fills for *L*. This process is repeated recursively for all vertices in the G(A) until all fills are identified.

An important point to consider that there may exist a matrix where the nonzero elements of are less than the nonzero elements of . The goal is to identify the permutation of matrix *A* that minimizes the number of fills in matrix *L.* Unfortunately, this is a very difficult problem (NP-complete) [3]. However, a heuristic algorithm called *minimum degree ordering algorithm* is widely used [4]. The algorithm doesn’t necessarily find the minimum but it will find *P* that generate with acceptable number of nonzero elements.

Applying the *minimum degree ordering and graph elimination algorithms* for identifying the *P* matrix and locating the nonzero elements in *L* required only once at the beginning of the optimizations and not required for each step.

**References:**

1. George, A., Liu, J., & Ng, E., 1994. *Computer Solution of Sparse Linear Systems*, pages 23-24 Academic Press, Orlando, Florida
2. Pothen and S. Toledo. Elimination structures in scientific computing. In D. Mehta and S. Sahni, editors, *Handbook on Data Structures and applications,* pages 59.1-59.29. CRC Press, 2004.
3. M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM Journal on Algebraic and Discrete Methods, 2 (1), pp. 77-79, 1981
4. W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proceedings of the IEEE, 55, pp. 1801–1809, 1967