Acta Crystallographica Section A
Foundations of Crystallography
ISSN 0108-7673

Received 22 May 2008
Accepted 15 July 2008

How to multiply a matrix of normal equations by an arbitrary vector using FFT

Boris V. Strokopytov

V. A. Engelhardt Institute of Molecular Biology, 32 Vavilova Street, 119991 Moscow, Russian Federation. Correspondence e-mail: strokop@gmail.com

This paper describes a novel algorithm for multiplying a matrix of normal equations by an arbitrary real vector using the fast Fourier transform technique. The algorithm allows full-matrix least-squares refinement of macromolecular structures without explicit calculation of the normal matrix. The resulting equations have been implemented in a new computer program, *FMLSQ*. A preliminary version of the program has been tested on several protein structures. The consequences for crystallographic refinement of macromolecules are discussed in detail.

1. Notations

In what follows, vectors are represented by their coordinates in columns and T stands for the transposition of vectors and matrices. Matrices and vectors are shown in bold.

$\otimes$	Convolution sign
$\|\bullet\|$	Vector norm
a_l, b_l	Atomic form-factor scattering constants
$B_i^{\rm iso}$	Isotropic atomic displacement factor
CPU	Central processing unit
Δx_i	Shift to x coordinate of ith atom
Δy_i	Shift to y coordinate of ith atom
Δz_i	Shift to z coordinate of ith atom
ΔB_i	Shift to isotropic atomic displacement parameter for ith atom
ΔO_i	Shift to atomic occupancy of ith atom
$\Delta\mathbf{p}$	Vector of parameter shifts: $(\Delta p_1, \Delta p_2, \ldots, \Delta p_{5N})^{\rm T} = (\Delta x_1, \Delta y_1, \ldots, \Delta O_N)^{\rm T}$
$\mathbf{D}$	Deorthogonalization three-by-three matrix
$f(\mathbf{p})$	Minimized function
$f_i^{\rm scatt}$	Form factor of ith atom
$F_{\rm c}$	Calculated structure factor
FFT	Fast Fourier transform
$\mathcal{F}^{-1}$	Complex Hermitian Fourier transform
$\mathbf{g}$	Gradient of minimized function $f(\mathbf{p})$
$\mathbf{H}$	Normal $5N$-by-$5N$ matrix
$H_1(x_i, x_j)$	H_1 normal matrix terms as defined by Agarwal (1978)
$[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m}$	Single entry vector of length $5N$: $(0, 0, \ldots, [\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m}, \ldots, 0)^{\rm T}$
$[\mathbf{H}\Delta\mathbf{p}]_x$	Scalar value equal to non-zero entry of $[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+1}$
$H_{\rm o}$	Orthogonalized index h: $\begin{pmatrix} H_{\rm o} \\ K_{\rm o} \\ L_{\rm o} \end{pmatrix} = \mathbf{D}^{\rm T} \begin{pmatrix} h \\ k \\ l \end{pmatrix}$
$\mathbf{I}$	Identity matrix
i, j	Atom indices
M	Size of normal matrix atom block; normally $M = 5$
m	Additional pointer to $\mathbf{H}\Delta\mathbf{p}$ vector entry; $m \le M$
ν	CPU time to evaluate one matrix–vector product
n	Number of iterations to solve one linear system of equations
N	Number of atoms in the asymmetric part of the unit cell
$N_{\rm sym}$	Number of symmetry operators for *true* space group
ngauss	Number of Gaussians used to approximate atomic scattering form factor
$\mathbf{O}$	Orthogonalization three-by-three matrix
$\varphi_{\rm c}$	Calculated phase in *true* space group
$\mathbf{p}_i$	Vector of parameters for ith atom: $\mathbf{p}_i = (x_i, y_i, z_i, B_i, O_i)^{\rm T}$
$\mathbf{q}$	Vector of quasi-occupancies: $\mathbf{q} = (q_1, q_2, \ldots, q_{5N})^{\rm T}$
$\mathbf{q}^{\rm mod}$	$5N$ vector of modified occupancies
ρ_x	Electron-density map
$\mathbf{r}$	Vector of position in real space
$\mathbf{r}_i$	Vector of position of ith atom in real space: $\mathbf{r}_i = (x_i, y_i, z_i)^{\rm T}$
$\Sigma_{5(i-1)+1}^{xx}$	Weighted sum of $H_1(x_i, x_j)$ or $H_2(x_i, x_j)$ terms defined by Agarwal (1978); during matrix–vector multiplication 25 such sums must be accumulated for H_1 terms and the same number of sums for H_2 terms
$\mathbf{s}$	Vector of position in reciprocal space: $\mathbf{s} = (h, k, l)$
$\|\mathbf{s}\|^2$	$4\sin^2\theta/\lambda^2$, squared distance in reciprocal space
$w(\mathbf{s})$	Spherically symmetric weighting function

2. Introduction

Full-matrix least-squares refinement of macromolecules has caused considerable interest during the past decade (Ten Eyck, 2003; Cowtan & Ten Eyck, 2000). The advantages of the least-squares method are known: fast convergence properties of the algorithm and the possibility to estimate the accuracy of refined parameters. Ten Eyck (1999) has also suggested that the full-matrix technique could be used for proper estimation of weighting schemes during the course of crystallographic refinement.

Though the method of full-matrix least-squares has been applied to refinement of small molecules for many years, it seems that full-matrix refinement of macromolecular structures remains out of reach. The reasons are obvious. Application of the full-matrix least-squares method requires construction of the following system of linear equations (see *e.g.* Ten Eyck, 1999):

$$\mathbf{H}\Delta\mathbf{p} = -\mathbf{g}. \tag{1}$$

A straightforward approach suggests that to obtain the parameter shifts $\Delta\mathbf{p}$ one needs to invert the matrix $\mathbf{H}$:

$$\Delta\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}. \tag{2}$$

This technique is implemented in many programs that deal with small molecules (Sheldrick, 2008). However, for proteins and other macromolecules calculation and storage of large matrices and especially their inversion may be prohibitively expensive in terms of both storage and CPU time. A simple calculation suggests that, for proteins larger than 2500 atoms in size, matrices containing hundreds of millions of elements must be constructed and inverted (assuming there are just four refinable parameters per atom). The situation will become even worse if we are going to refine coordinates and anisotropic displacement parameters (nine refinable parameters per atom) or deal with much larger structures. Therefore, most of the modern macromolecular crystallographic refinement programs like *CNS* (Brünger *et al.*, 1998), *REFMAC5* (Murshudov *et al.*, 1997) and *TNT* (Tronrud, 1997) use a rather rough approximation of the normal matrix (diagonal or block diagonal). This drastically reduces the amount of storage needed for the normal matrix and in addition its inversion becomes almost trivial. However, convergence of these algorithms may become slow at the end of the refinement and error analysis remains a difficult problem.

Clearly, most macromolecular structures would benefit from application of full-matrix least squares during the last stages of refinement; model parameters are expected to have more accurate values at the end of refinement (especially displacement and occupancy parameters) compared with the values obtained by the method of function minimization by conjugate gradients, since interdependencies between model parameters are taken into account. Analysis of the main diagonal of the normal matrix inverse may be valuable for the estimation of proper weighting schemes. Eigenvector analysis of the normal matrix could be applied to find poorly or incorrectly defined regions of protein structures (Cowtan & Ten Eyck, 2000). Even a remote possibility of estimating standard uncertainties for model parameters of larger proteins through the method of least squares seems to be of considerable interest. However, it is obvious that a completely different approach is necessary to achieve these goals.

The most intriguing question to ask is whether it is possible, at least in principle, to avoid calculation of the normal matrix altogether. A positive answer comes from two works published in the early 1950s, by Hestenes & Stiefel (1952) and Lanczos (1952). These authors suggested a method (based on conjugate gradients) for solution of linear systems $\mathbf{Ax} = \mathbf{b}$ which does not require explicit matrix inversion. Since then various modifications of this method have been developed. A brief overview of these approaches to the solution of linear systems was given by Urzhumtsev & Lunin (2001). For us, the most important aspect of these algorithms is that one does not need to calculate explicitly all n^2 components of the matrix $\mathbf{H}$ but only its product by a vector, *i.e.* just n values.

Application of the fast differentiation algorithm (*FDA*; Kim *et al.*, 1984) to crystallographic refinement has been performed by Urzhumtsev & Lunin (2001) and Lunin & Urzhumtsev (1985). On this basis, these authors have suggested that an algorithm for normal matrix–vector multiplication must exist and even predicted the numerical complexity of this operation. However, to the best of our knowledge, this algorithm remained undiscovered for quite some time.

This paper describes a novel technique for calculation of the normal matrix–vector product which avoids explicit calculation and storage of the normal matrix in computer memory. As shown below, this technique allows full-matrix least-squares refinement of proteins of virtually any size.

2.1. What was known: Agarwal's result

The methods used to calculate the gradient vector and matrix of normal equations have been given by Agarwal (1978). We briefly review his method since it was instrumental for the development of our matrix–vector product algorithm. This will allow us to set up appropriate notation as well.

Let us give some basic formulas and definitions. In this paper we are only concerned with the X-ray part of the normal equations. The minimized function is

$$f(\mathbf{p}) = \sum_{\mathbf{s}} w(\mathbf{s})E^2(\mathbf{s},\mathbf{p}), \tag{3}$$

$$E(\mathbf{s}) = |F_{\rm o}(\mathbf{s})| - |F_{\rm c}(\mathbf{s},\mathbf{p})|. \tag{4}$$

If we define now

$$g_i(\mathbf{s}) = f_i^{\rm scatt}(\mathbf{s})\exp\left(-B_i^{\rm iso}|\mathbf{s}|^2/4\right) \tag{5}$$

to simplify the expressions below, the calculated structure factor (omitting vector $\mathbf{p}$) may be written as

$$F_{\rm c}(\mathbf{s}) = \sum_{i=1}^{N} O_i g_i(\mathbf{s})\exp(2\pi i\mathbf{s}\mathbf{r}_i). \tag{6}$$

The expression for the gradient of the x coordinate for the ith atom obtained by Agarwal is

$$\frac{\partial f(\mathbf{p})}{\partial x_i} = 2\sum_{\mathbf{s}} 2\pi i h\, w(\mathbf{s})\, E(\mathbf{s})\, O_i\, g_i(\mathbf{s}) \exp[i\varphi_c(\mathbf{s})] \exp(-2\pi i \mathbf{s}\mathbf{r}_i). \tag{7}$$

This corresponds (with minor modifications) to equation (27) from the original Agarwal (1978) paper. Similar expressions were derived by Agarwal for other types of model parameters. In addition, Agarwal was able to demonstrate that the normal matrix could be represented by a sum of two matrices:

$$\mathbf{H} = \mathbf{H}_1 + \mathbf{H}_2. \tag{8}$$

The normal matrix (or, in other words, the matrix of second derivatives after linearization) has the following form:

$$\frac{\partial^2 f}{\partial \mathbf{p}_i \partial \mathbf{p}_j} = 2\sum_{\mathbf{s}} w(\mathbf{s}) \left[\frac{\partial |F_c(\mathbf{s})|}{\partial \mathbf{p}_i}\right] \left[\frac{\partial |F_c(\mathbf{s})|}{\partial \mathbf{p}_j}\right]^{\rm T}, \tag{9}$$

where $\mathbf{p}_i$ is the vector of parameters describing the ith atom. The equivalent compact form has been given by Tronrud (1999). Here we present another version of the compact form with modifications that correspond directly to the result obtained by Agarwal (1978) with some corrections by Lifchitz (1986). Separating atomic occupancies from atomic scattering factors[1] we may write

$$\begin{aligned}\frac{\partial^2 f}{\partial \mathbf{p}_i \partial \mathbf{p}_j} = {}& \sum_{\mathbf{s}} w(\mathbf{s}) g_i(\mathbf{s}) g_j(\mathbf{s}) \\ & \times \begin{pmatrix} 4\pi^2 h^2 O_i O_j & 4\pi^2 hk O_i O_j & 4\pi^2 hl O_i O_j & -\pi i h \frac{s^2}{2} O_i O_j & 2\pi i h O_i \\ 4\pi^2 hk O_i O_j & 4\pi^2 k^2 O_i O_j & 4\pi^2 kl O_i O_j & -\pi i k \frac{s^2}{2} O_i O_j & 2\pi i k O_i \\ 4\pi^2 hl O_i O_j & 4\pi^2 kl O_i O_j & 4\pi^2 l^2 O_i O_j & -\pi i l \frac{s^2}{2} O_i O_j & 2\pi i l O_i \\ \pi i h \frac{s^2}{2} O_i O_j & \pi i k \frac{s^2}{2} O_i O_j & \pi i l \frac{s^2}{2} O_i O_j & \frac{s^4}{16} O_i O_j & \frac{-s^2}{4} O_i \\ -2\pi i h O_j & -2\pi i k O_j & -2\pi i l O_j & \frac{-s^2}{4} O_j & 1 \end{pmatrix} \\ & \times \exp[2\pi i \mathbf{s}(\mathbf{r}_i - \mathbf{r}_j)] - \sum_{\mathbf{s}} w(\mathbf{s}) g_i(\mathbf{s}) g_j(\mathbf{s}) \exp[2i\varphi_c(\mathbf{s})] \\ & \times \begin{pmatrix} 4\pi^2 h^2 O_i O_j & 4\pi^2 hk O_i O_j & 4\pi^2 hl O_i O_j & -\pi i h \frac{s^2}{2} O_i O_j & 2\pi i h O_i \\ 4\pi^2 hk O_i O_j & 4\pi^2 k^2 O_i O_j & 4\pi^2 kl O_i O_j & -\pi i k \frac{s^2}{2} O_i O_j & 2\pi i k O_i \\ 4\pi^2 hl O_i O_j & 4\pi^2 kl O_i O_j & 4\pi^2 l^2 O_i O_j & -\pi i l \frac{s^2}{2} O_i O_j & 2\pi i l O_i \\ -\pi i h \frac{s^2}{2} O_i O_j & -\pi i k \frac{s^2}{2} O_i O_j & -\pi i l \frac{s^2}{2} O_i O_j & \frac{-s^4}{16} O_i O_j & \frac{s^2}{4} O_i \\ 2\pi i h O_j & 2\pi i k O_j & 2\pi i l O_j & \frac{s^2}{4} O_j & -1 \end{pmatrix} \\ & \times \exp[-2\pi i \mathbf{s}(\mathbf{r}_i + \mathbf{r}_j)]. \end{aligned} \tag{10}$$

Equation (10) corresponds to one normal matrix block defining the interaction of two atoms with indices i and j.

Inspection of equation (10) reveals that the normal matrix is composed of 25 groups/sets of coefficients for H_1 matrix terms and the same number of sets for H_2 terms. We are ready now to derive formulae for the matrix–vector product.

[1] Careful tracking of atomic occupancies is important for matrix–vector product derivation.

3. Matrix–vector product derivation

Following the notation of Agarwal for the normal matrix terms, multiplication of the normal matrix $\mathbf{H}$ by a column vector $\Delta\mathbf{p}$ requires accumulation of the following sums:

$$\mathbf{H}\Delta\mathbf{p} = \sum_l H_{kl}\Delta p_l = \sum_l H_{1kl}\Delta p_l + \sum_l H_{2kl}\Delta p_l, \tag{11}$$

where $k = 1, 2, \ldots, 5N$ and $l = 1, 2, \ldots, 5N$.

To start the derivation let us first consider the $H_1(x_i, x_j)$ normal matrix term corresponding to the matrix element (1,1) in the first (upper) matrix in equation (10):

$$H_1(x_i, x_j) = \sum_{\mathbf{s}} w(\mathbf{s})\, 4\pi^2 h^2 O_i O_j g_i(\mathbf{s}) g_j(\mathbf{s}) \exp[2\pi i \mathbf{s}(\mathbf{r}_i - \mathbf{r}_j)], \tag{12}$$

where i and j are atom indices, $i, j = 1, 2, \ldots, N$.

How frequently can we find those $H_1(x_i, x_j)$ terms in the H_1 part of the normal matrix? Inspection of the entries in equation (10) suggests that there are N^2 entries of this type in N rows of the $5N$-by-$5N$ $\mathbf{H}_1$ matrix. They are positioned at columns with indices $5(j-1)+1$ and rows with indices $5(i-1)+1$. Thus only N (out of $5N$) components of $\Delta\mathbf{p}$ in each row with indices $5(i-1)+1$ will participate in the following summation:[2]

$$\Sigma_{5(i-1)+1}^{H_1(x_i,x_j)} = \sum_{j=1}^{N} H_1(x_i, x_j)\Delta p_{5(j-1)+1}. \tag{13}$$

To simplify the notation we shall replace the upper index $H_1(x_i, x_j)$ with xx, and the above equation reads now as

$$\Sigma_{5(i-1)+1}^{xx} = \sum_{j=1}^{N} H_1(x_i, x_j)\Delta p_{5(j-1)+1}. \tag{14}$$

The lower index in $\Sigma_{5(i-1)+1}^{xx}$ means that this summation is valid only for rows with indices $k = 1, 6, \ldots, 5(N-1)+1$. Obviously, similar expressions can be written for each of the 25 terms in equation (10). For summation of $H_1(y_i, x_j)$ terms we will use the upper index yx *etc*. The same type of indexing will be used for H_2 terms. In total 25 various upper indices will be used, which can be conveniently represented by the following symbolic matrix:

$$\begin{pmatrix} xx & xy & xz & xB & xO \\ yx & yy & yz & yB & yO \\ zx & zy & zz & zB & zO \\ Bx & By & Bz & BB & BO \\ Ox & Oy & Oz & OB & OO \end{pmatrix}. \tag{15}$$

It is clear that matrix (15) has direct correspondence to equation (10). Decomposing the $\mathbf{H}_1\Delta\mathbf{p}$ product into five equal sets (according to the size of atomic block) we may write

$$\begin{aligned}[\mathbf{H}_1\Delta\mathbf{p}]_{5(i-1)+1} = {}& \Sigma_{5(i-1)+1}^{xx} + \Sigma_{5(i-1)+1}^{xy} + \Sigma_{5(i-1)+1}^{xz} \\ & + \Sigma_{5(i-1)+1}^{xB} + \Sigma_{5(i-1)+1}^{xO}, \end{aligned} \tag{16}$$

[2] In practice the situation is a bit more complicated since not all atomic blocks may be equal in size and each atom may have a different number of refinable parameters. This problem is addressed at the programming level by creating appropriate pointers to normal matrix atomic blocks or vector entries. Here, for simplicity, we assume that all atoms have the same number of refinable parameters and all atomic blocks defined by equation (10) are equal.

$$[\mathbf{H}_1\Delta\mathbf{p}]_{5(i-1)+2} = \Sigma^{yx}_{5(i-1)+2} + \Sigma^{yy}_{5(i-1)+2} + \Sigma^{yz}_{5(i-1)+2} + \Sigma^{yB}_{5(i-1)+2} + \Sigma^{yO}_{5(i-1)+2}, \quad (17)$$

$$[\mathbf{H}_1\Delta\mathbf{p}]_{5(i-1)+3} = \Sigma^{zx}_{5(i-1)+3} + \Sigma^{zy}_{5(i-1)+3} + \Sigma^{zz}_{5(i-1)+3} + \Sigma^{zB}_{5(i-1)+3} + \Sigma^{zO}_{5(i-1)+3}, \quad (18)$$

$$[\mathbf{H}_1\Delta\mathbf{p}]_{5(i-1)+4} = \Sigma^{Bx}_{5(i-1)+4} + \Sigma^{By}_{5(i-1)+4} + \Sigma^{Bz}_{5(i-1)+4} + \Sigma^{BB}_{5(i-1)+4} + \Sigma^{BO}_{5(i-1)+4}, \quad (19)$$

$$[\mathbf{H}_1\Delta\mathbf{p}]_{5i} = \Sigma^{Ox}_{5i} + \Sigma^{Oy}_{5i} + \Sigma^{Oz}_{5i} + \Sigma^{OB}_{5i} + \Sigma^{OO}_{5i}. \quad (20)$$

Substituting $\mathbf{H}_2$ for $\mathbf{H}_1$, exactly the same equations can be written in terms of H_2. To get the full $\mathbf{H}\Delta\mathbf{p}$ product we need to sum the H_1 and H_2 normal matrix terms:

$$[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m} = [\mathbf{H}_1\Delta\mathbf{p}]_{5(i-1)+m} + [\mathbf{H}_2\Delta\mathbf{p}]_{5(i-1)+m} \quad (21)$$

for $i = 1, 2 \ldots, N$ and $m = 1, 2, \ldots, 5$. Thus we have a general picture of further calculations. Let us go back to the details.

Explicit multiplication for equation (14) using equation (12) gives, in space group $P1$,

$$\Sigma^{xx}_{5(i-1)+1} = \sum_{j=1}^{N} \Delta p_{5(j-1)+1} \sum_{\mathbf{s}} w(\mathbf{s})\, 4\pi^2 h^2 O_i O_j g_i(\mathbf{s}) g_j(\mathbf{s}) \times \exp[2\pi i \mathbf{s}(\mathbf{r}_i - \mathbf{r}_j)], \quad (22)$$

where $i = 1, 2, \ldots, N$. Changing the order of the summation (to separate items that do not depend on index j), splitting the exponent into two parts and rearranging some terms one can easily obtain

$$\Sigma^{xx}_{5(i-1)+1} = \sum_{\mathbf{s}} O_i g_i(\mathbf{s}) 4\pi^2 h^2 w(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i) \times \sum_{j=1}^{N} \Delta p^{x}_{5(j-1)+1} O_j g_j(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_j). \quad (23)$$

Summation of all matrix entries (weighted by vector $\Delta\mathbf{p}$ components), for example, for the first row of matrix $\mathbf{H}_1$, requires a split of the vector $\Delta\mathbf{p}$ into five equal sets of components:[3]

$$\Delta\mathbf{p} = \begin{cases} \Delta p^{x}_{5(j-1)+1}, \\ \Delta p^{y}_{5(j-1)+2}, \\ \Delta p^{z}_{5(j-1)+3}, \quad j = 1, 2, \ldots, N. \\ \Delta p^{B}_{5(j-1)+4}, \\ \Delta p^{O}_{5j} \end{cases} \quad (24)$$

The sets defined by equation (24) can be applied during multiplication for all other matrix rows. Summation for the rows with indices $k = 1, 6, \ldots, 5(N-1)+1$ for the $\mathbf{H}_1$ matrix can now be written using equation (10) as a look-up table. Applying the transformations we used to derive equation (23) to each set of coefficients and summing the $\mathbf{H}_1\Delta\mathbf{p}$ and $\mathbf{H}_2\Delta\mathbf{p}$ products according to equation (21), it can be shown that

[3] Upper indices x, y, z, B, O have been introduced to reflect the fact that $\Delta p^{x}_{5(j-1)+1} = \Delta x_j$, $\Delta p^{y}_{5(j-1)+2} = \Delta y_j$ *etc.*

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+1} = &\sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 h^2 w(\mathbf{s})\, E^{x}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 hkw(\mathbf{s})\, E^{y}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 hlw(\mathbf{s})\, E^{z}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, \pi ih(s^2/2)w(\mathbf{s})\, E^{B}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 2\pi ihw(\mathbf{s})\, E^{O}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i), \end{aligned} \quad (25)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+2} = &\sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 hkw(\mathbf{s})\, E^{x}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 k^2 w(\mathbf{s})\, E^{y}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 klw(\mathbf{s})\, E^{z}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, \pi ik(s^2/2)w(\mathbf{s})E^{B}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 2\pi ikw(\mathbf{s})E^{O}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i), \end{aligned} \quad (26)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+3} = &\sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 hlw(\mathbf{s})E^{x}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 klw(\mathbf{s})E^{y}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 l^2 w(\mathbf{s})E^{z}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, \pi il(s^2/2)w(\mathbf{s})E^{B}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 2\pi ilw(\mathbf{s})E^{O}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i), \end{aligned} \quad (27)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+4} = &-\sum_{\mathbf{s}} O_i g_i(\mathbf{s})\pi ih(s^2/2)w(\mathbf{s})E^{x}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\pi ik(s^2/2)w(\mathbf{s})E^{y}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} O_i g_i(\mathbf{s})\pi il(s^2/2)w(\mathbf{s})E^{z}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} O_i g_i(\mathbf{s})(s^4/16)w(\mathbf{s})E^{B}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} O_i g_i(\mathbf{s})(s^2/4)w(\mathbf{s})E^{O}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i), \end{aligned} \quad (28)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5i} = &\sum_{\mathbf{s}} g_i(\mathbf{s})2\pi ihw(\mathbf{s})E^{x}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} g_i(\mathbf{s})2\pi ikw(\mathbf{s})E^{y}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} g_i(\mathbf{s})2\pi ilw(\mathbf{s})E^{z}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &- \sum_{\mathbf{s}} g_i(\mathbf{s})(s^2/4)w(\mathbf{s})E^{B}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i) \\ &+ \sum_{\mathbf{s}} g_i(\mathbf{s})w(\mathbf{s})E^{O}(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i), \end{aligned} \quad (29)$$

where

$$E^{x}(\mathbf{s}) = F^{x}_{c}(\mathbf{s}) - F^{x*}_{c}(\mathbf{s}) \exp[2i\varphi_c(\mathbf{s})], \quad (30)$$

$$E^{y}(\mathbf{s}) = F^{y}_{c}(\mathbf{s}) - F^{y*}_{c}(\mathbf{s}) \exp[2i\varphi_c(\mathbf{s})], \quad (31)$$

$$E^{z}(\mathbf{s}) = F^{z}_{c}(\mathbf{s}) - F^{z*}_{c}(\mathbf{s}) \exp[2i\varphi_c(\mathbf{s})], \quad (32)$$

$$E^B(\mathbf{s}) = F_c^B(\mathbf{s}) + F_c^{B*}(\mathbf{s})\exp[2i\varphi_c(\mathbf{s})], \tag{33}$$

$$E^O(\mathbf{s}) = F_c^O(\mathbf{s}) + F_c^{O*}(\mathbf{s})\exp[2i\varphi_c(\mathbf{s})]. \tag{34}$$

The F_c values are calculated according to

$$\begin{aligned} F_c^{x*}(\mathbf{s}) &= \sum_{j=1}^{N} q^x_{5(j-1)+1} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j) \\ &= \sum_{j=1}^{N} \Delta p^x_{5(j-1)+1} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j), \end{aligned} \tag{35}$$

$$\begin{aligned} F_c^{y*}(\mathbf{s}) &= \sum_{j=1}^{N} q^y_{5(j-1)+2} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j) \\ &= \sum_{j=1}^{N} \Delta p^y_{5(j-1)+2} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j), \end{aligned} \tag{36}$$

$$\begin{aligned} F_c^{z*}(\mathbf{s}) &= \sum_{j=1}^{N} q^z_{5(j-1)+3} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j) \\ &= \sum_{j=1}^{N} \Delta p^z_{5(j-1)+3} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j), \end{aligned} \tag{37}$$

$$\begin{aligned} F_c^{B*}(\mathbf{s}) &= \sum_{j=1}^{N} q^B_{5(j-1)+4} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j) \\ &= \sum_{j=1}^{N} \Delta p^B_{5(j-1)+4} O_j\, g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j), \end{aligned} \tag{38}$$

$$\begin{aligned} F_c^{O*}(\mathbf{s}) &= \sum_{j=1}^{N} q^O_{5j} g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j) \\ &= \sum_{j=1}^{N} \Delta p^O_{5j} g_j(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_j). \end{aligned} \tag{39}$$

Therefore, five sets of modified occupancies must be formed to calculate the corresponding F_c parameters:

$$\mathbf{q}^{\rm mod} = \begin{cases} q^x_{5(j-1)+1}O_j \\ q^y_{5(j-1)+2}O_j \\ q^z_{5(j-1)+3}O_j \quad j = 1, 2, \ldots, N \\ q^B_{5(j-1)+4}O_j \\ q^O_{5j} \end{cases} \tag{40}$$

and

$$\Delta\mathbf{p} \equiv \mathbf{q} = (q_1, q_2, \ldots, q_{5N})^{\rm T}. \tag{41}$$

During the simplification process we assumed that Friedel's law holds and the following two important identities have been used:

$$\sum_{\mathbf{s}} F_c^*(\mathbf{s})\exp(2\pi i\mathbf{s}\mathbf{r}_i) = \sum_{\mathbf{s}} F_c(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_i) \tag{42}$$

and

$$\sum_{\mathbf{s}} iF_c^*(\mathbf{s})\exp(2\pi i\mathbf{s}\mathbf{r}_i) = -\sum_{\mathbf{s}} iF_c(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_i). \tag{43}$$

All single entry vectors $[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m}$ can be merged readily into the final $\mathbf{H}\Delta\mathbf{p}$ product:

$$\mathbf{H}\Delta\mathbf{p} = \sum_{m=1}^{5}\sum_{i=1}^{N}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m}. \tag{44}$$

Obviously, vector $\mathbf{H}\Delta\mathbf{p}$ has no empty entries now.

This completes the list of all the terms necessary to calculate the desired matrix–vector product in the $P1$ space group.

A more detailed derivation of the matrix–vector product can be found in the supplementary material[4] for this paper.

3.1. Using FFT

To see how the FFT algorithm could be applied to calculation of the matrix–vector product let us first consider as an example the first sum in equation (25):

$$\sum_{\mathbf{s}} O_i g_i(\mathbf{s})\, 4\pi^2 h^2 w(\mathbf{s}) E^x(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}_i). \tag{45}$$

This expression can be viewed as a product of two functions and according to the convolution theorem can be evaluated using convolution of the map calculated with the following coefficients:

$$\rho_{xx}(\mathbf{r}) = \sum_{\mathbf{s}} 4\pi^2 h^2 w(\mathbf{s}) E^x(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}), \tag{46}$$

with the Fourier transform[5] of $O_i g_i(\mathbf{s})\exp(2\pi i\mathbf{s}\mathbf{r}_i)$:

$$\rho_i^{\rm atom}(\mathbf{r} - \mathbf{r}_i) = O_i \sum_{l}^{\rm ngauss} a_l \left(\frac{4\pi}{B_i^{\rm iso} + b_l}\right)^{3/2} \exp\left(-\frac{4\pi^2 r^2}{B_i^{\rm iso} + b_l}\right), \tag{47}$$

where $r^2 = ||\mathbf{r} - \mathbf{r}_i||_2^2$ in an orthogonal coordinate frame. $F_c^x(\mathbf{s})$ are calculated according to equation (35). $E^x(\mathbf{s})$ are given by equation (30).

Inspection of equation (25) suggests that the evaluation of the whole product for rows with indices $5(i-1)+1$ needs a more complex map, which would combine the following five sets of coefficients:

$$\begin{aligned} \rho_x(\mathbf{r}) &= \rho_{xx}(\mathbf{r}) + \rho_{xy}(\mathbf{r}) + \rho_{xz}(\mathbf{r}) + \rho_{xB}(\mathbf{r}) + \rho_{xO}(\mathbf{r}) \\ &= \sum_{\mathbf{s}} 4\pi^2 h^2 w(\mathbf{s}) E^x(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}) \\ &\quad + \sum_{\mathbf{s}} 4\pi^2 hk w(\mathbf{s}) E^y(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}) \\ &\quad + \sum_{\mathbf{s}} 4\pi^2 hl w(\mathbf{s}) E^z(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}) \\ &\quad + \sum_{\mathbf{s}} \pi i h(s^2/2) w(\mathbf{s}) E^B(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}) \\ &\quad - \sum_{\mathbf{s}} 2\pi i h w(\mathbf{s}) E^O(\mathbf{s})\exp(-2\pi i\mathbf{s}\mathbf{r}). \end{aligned} \tag{48}$$

During the next step this combined map has to be convoluted with the atomic densities of all atoms producing a partial matrix–vector product for rows $5(i-1)+1$, $i = 1, 2, \ldots, N$.

The four other maps must be calculated in order to complete the evaluation of the full matrix–vector product:

[4] Supplementary material for this paper is available from the IUCr electronic archives (Reference: ZM5045). Services for accessing these archives are described at the back of the journal.

[5] When calculating the convolution with the map defined by equation (52) the O_i term has to be omitted. See also equations (29) and (57).

$$\begin{aligned}\rho_y(\mathbf{r}) &= \rho_{yx}(\mathbf{r}) + \rho_{yy}(\mathbf{r}) + \rho_{yz}(\mathbf{r}) + \rho_{yB}(\mathbf{r}) + \rho_{yO}(\mathbf{r}) \\ &= \sum_{\mathbf{s}} 4\pi^2 hkw(\mathbf{s})E^x(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} 4\pi^2 k^2 w(\mathbf{s})E^y(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} 4\pi^2 klw(\mathbf{s})E^z(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} \pi ik(s^2/2)w(\mathbf{s})E^B(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad - \sum_{\mathbf{s}} 2\pi ikw(\mathbf{s})E^O(\mathbf{s})\exp(-2\pi i\mathbf{sr}), \end{aligned} \qquad (49)$$

$$\begin{aligned}\rho_z(\mathbf{r}) &= \rho_{zx}(\mathbf{r}) + \rho_{zy}(\mathbf{r}) + \rho_{zz}(\mathbf{r}) + \rho_{zB}(\mathbf{r}) + \rho_{zO}(\mathbf{r}) \\ &= \sum_{\mathbf{s}} 4\pi^2 hlw(\mathbf{s})E^x(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} 4\pi^2 klw(\mathbf{s})E^y(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} 4\pi^2 l^2 w(\mathbf{s})E^z(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} \pi il(s^2/2)w(\mathbf{s})E^B(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad - \sum_{\mathbf{s}} 2\pi ilw(\mathbf{s})E^O(\mathbf{s})\exp(-2\pi i\mathbf{sr}), \end{aligned} \qquad (50)$$

$$\begin{aligned}\rho_B(\mathbf{r}) &= \rho_{Bx}(\mathbf{r}) + \rho_{By}(\mathbf{r}) + \rho_{Bz}(\mathbf{r}) + \rho_{BB}(\mathbf{r}) + \rho_{BO}(\mathbf{r}) \\ &= -\sum_{\mathbf{s}} \pi ih(s^2/2)w(\mathbf{s})E^x(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad - \sum_{\mathbf{s}} \pi ik(s^2/2)w(\mathbf{s})E^y(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad - \sum_{\mathbf{s}} \pi il(s^2/2)w(\mathbf{s})E^z(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} (s^4/16)w(\mathbf{s})E^B(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad - \sum_{\mathbf{s}} (s^2/4)w(\mathbf{s})E^O(\mathbf{s})\exp(-2\pi i\mathbf{sr}), \end{aligned} \qquad (51)$$

$$\begin{aligned}\rho_O(\mathbf{r}) &= \rho_{Ox}(\mathbf{r}) + \rho_{Oy}(\mathbf{r}) + \rho_{Oz}(\mathbf{r}) + \rho_{OB}(\mathbf{r}) + \rho_{OO}(\mathbf{r}) \\ &= \sum_{\mathbf{s}} 2\pi ihw(\mathbf{s})E^x(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} 2\pi ikw(\mathbf{s})E^y(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} 2\pi ilw(\mathbf{s})E^z(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad - \sum_{\mathbf{s}} (s^2/4)w(\mathbf{s})E^B(\mathbf{s})\exp(-2\pi i\mathbf{sr}) \\ &\quad + \sum_{\mathbf{s}} w(\mathbf{s})E^O(\mathbf{s})\exp(-2\pi i\mathbf{sr}). \end{aligned} \qquad (52)$$

The F_c values (and thus E^x, E^y *etc.*) can be generated by means of conventional FFT methods through density generation routines, which have to be modified somewhat to allow arbitrarily large or small occupancies on input. Note that the quasi-occupancies $\mathbf{q}$ defined by equation (41) may assume negative values.

In total, 25 sets of coefficients have to be calculated and appropriately combined to produce the five different maps defined by equations (48)–(52). Convolution of each map with the atomic densities of N atoms will produce $5N$ single entry vectors $[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m}$ of length $5N$. Omitting some constant multipliers we may write out the final result:

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+1} &= \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\otimes\rho_x(\mathbf{r}) \\ &= \sum_{\mathbf{r}} \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\rho_x(\mathbf{r}), \end{aligned} \qquad (53)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+2} &= \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\otimes\rho_y(\mathbf{r}) \\ &= \sum_{\mathbf{r}} \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\rho_y(\mathbf{r}), \end{aligned} \qquad (54)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+3} &= \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\otimes\rho_z(\mathbf{r}) \\ &= \sum_{\mathbf{r}} \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\rho_z(\mathbf{r}), \end{aligned} \qquad (55)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+4} &= \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\otimes\rho_B(\mathbf{r}) \\ &= \sum_{\mathbf{r}} \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\rho_B(\mathbf{r}), \end{aligned} \qquad (56)$$

$$\begin{aligned}[\mathbf{H}\Delta\mathbf{p}]_{5i} &= (1/O_i)\rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\otimes\rho_O(\mathbf{r}) \\ &= (1/O_i)\sum_{\mathbf{r}} \rho_i^{\rm atom}(\mathbf{r}-\mathbf{r}_i)\rho_O(\mathbf{r}). \end{aligned} \qquad (57)$$

In the final step the $5N$ single entry vectors $[\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+m}$ can be merged into one vector of length $5N$ using equation (44), which corresponds to the desired $\mathbf{H}\Delta\mathbf{p}$ product.

The symmetries of the maps defined by equations (48)–(52) are unknown. Therefore it is advised to calculate coefficients for these maps in the $P1$ space group, noting that the reciprocal-space operations are relatively cheap in terms of CPU time. Further research in this area is required.

In Appendix *A* we provide some information about the calculation of the matrix–vector product in arbitrary space groups.

Derivation of the matrix–vector product can be achieved using several other approaches. Appendix *B* contains information about other possibilities to derive the full normal matrix–vector product.

Several other types of vector–matrix products can be easily calculated *via* FFT (see Appendix *C*).

3.2. Summarizing steps to calculate the normal matrix–vector product

The following steps have to be carried out to calculate the matrix–vector product $\mathbf{H}\Delta\mathbf{p}$:

(1) Generate five density maps for the model structure using the five sets of occupancies defined by equation (40).

(2) Evaluate five sets of structure factors defined by equation (35)–(39) using real FFT.

(3) Calculate coefficients for five maps in the $P1$ space group, equations (48)–(52).

(4) Apply symmetry operators in reciprocal space to obtain the matrix–vector product in the *true* space group (see Appendix *A*).

(5) Use complex Hermitian FFT to calculate the maps given by equations (48)–(52).

(6) Perform the convolution step for the five maps defined by equations (53)–(57).

(7) Assemble the final matrix–vector product [equation (44)].

A nice feature of this algorithm is that the most time-consuming steps – density generation and the convolution steps – may be easily parallelized (and that might be important if several CPUs are available). 80–90% of the CPU time is spent during the real-space operations.

3.3. Limitations of the method and the normal matrix inversion problem

The largest limitation of the method resides in its very nature. Individual matrix elements are not explicitly accessible. For example, calculation of all matrix elements requires $5N$ implicit multiplications using

$$\mathbf{H}_j = \mathbf{H}\mathbf{e}_j, \tag{58}$$

where $\mathbf{e}_j = (0, 0, \ldots, 1, \ldots, 0)^{\rm T}$, 1 is in the jth position and $\mathbf{H}_j$ denotes the jth column of the $\mathbf{H}$ matrix. This was pointed out by Urzhumtsev & Lunin (2001).

It could be added to this observation that explicit inversion of the whole normal matrix for larger proteins remains difficult (though formally possible). A straightforward approach which avoids direct calculation of the normal matrix using equation (58) suggests that we need to solve $5N$ linear systems to obtain the inverted matrix $\mathbf{H}^{-1}$:

$$\mathbf{H}\mathbf{X} = \mathbf{I}, \tag{59}$$

which is rather expensive in terms of CPU time, even for protein structures of medium size. (When all systems have been solved we obviously have $\mathbf{X} = \mathbf{H}^{-1}$.) The overall time $T_{\rm inv}$ for implicit matrix inversion using this method can be estimated as

$$T_{\rm inv} \simeq 5N\nu n, \tag{60}$$

where ν is the time required to evaluate one matrix–vector product and n is the average number of iterations required to solve one linear system. However, assuming that we can obtain a solution of the single linear system in a fixed number of iterations,[6] $T_{\rm inv}$ will scale as $O(N^2)$ since ν is roughly proportional to N. Note that for very large proteins we may choose to calculate selected columns only[7] (and hence corresponding diagonal elements) of the inverse matrix $\mathbf{H}^{-1}$ using this method.

For smaller proteins (around or less than 3000 atoms in the asymmetric part of the unit cell), another approach to normal matrix inversion is possible (if one really needs to obtain all diagonal elements of the matrix inverse). Equation (58) could be used to calculate matrix $\mathbf{H}$ explicitly and, as pointed out by an anonymous referee, we may use an efficient implementation of the *LAPACK* library (Anderson *et al.*, 1999) to obtain matrix $\mathbf{H}^{-1}$ (*e.g.* Intel *MKL* library). This approach might be more efficient than solution of $5N$ linear systems since the implicit normal matrix–vector multiplication is a relatively slow operation. However, comparison of both methods for smaller structures goes beyond the scope of this article. Clearly, for larger macromolecules the size of matrix $\mathbf{H}$ grows pretty quickly, and it may be too big to fit into computer memory at a certain point. Therefore we believe that iterative methods based on the matrix–vector algorithm are the only viable alternative for large proteins (*e.g.* 10 000 atoms or more).

In subsequent papers the problem of matrix inversion will be considered in more detail because of its importance for the calculation of s.u. values of model parameters. Note that explicit normal matrix inversion is needed for error analysis only and should be performed only once at the end of crystallographic refinement.

Limitations mentioned by Tronrud (1999) will also apply to the matrix–vector product algorithm.

3.4. Advantages

On a positive note, one cycle of least-squares refinement will require the CPU time $T_{\rm lsq} \ll T_{\rm inv}$:

$$T_{\rm lsq} \simeq \nu n. \tag{61}$$

Assuming that one matrix–vector product requires one minute of CPU time for a medium-sized protein and the number of iterations is about 30, it follows that one cycle of full-matrix refinement will take just about half an hour, which is quite acceptable in practice (see §4). Preconditioning of the normal matrix may be used to reduce the number of iterations when solving a linear system. This will speed up convergence and in addition improves the accuracy of the solution. Normally $n \ll 5N$, even in those cases when one uses just the simplest diagonal preconditioner. A brief overview of the theory of preconditioners is given in Appendix *D*.

No explicit storage of the normal matrix is required. However, memory requirements are still relatively high, since up to five electron-density maps have to be held in computer memory simultaneously. (It is possible to use one map only, gradually accumulating the matrix–vector product, but this will slow down the matrix–vector product calculations considerably.)

The most important fact is that virtually all linear algebra algorithms remain accessible and can be easily implemented using the matrix–vector product algorithm.

3.5. Some technical details of *FMLSQ*

The above equations have been implemented in a new computer program, *FMLSQ* (full-matrix least-squares) using the Fortran95 language. For speedy calculation of the matrix–vector products the program was arranged to keep all five maps in computer memory. To minimize the required memory and, on the other hand, produce accurate results we rely on the method developed by Navaza (2002).

The routines for the density generation and convolution step were written from scratch to handle up to five maps simultaneously and allow arbitrary values for occupancies on input, which is essential for correct functioning of the algo-

[6] Actually the dependence between the number of iterations and the matrix size is a complex matter. The speed of convergence and the number of iterations depend on the spectrum of the normal matrix and the choice of preconditioner. See Appendix *D*.

[7] Solving the linear system $\mathbf{H}\mathbf{x} = \mathbf{e}_j$ for each desired parameter with index j.

Table 1
Basic crystallographic data for four structures.

Protein	Resolution (Å)	No. of reflections	No. of atoms	Space group	R value†
α-Conotoxin PnIB	1.1	4358	136	$P2_12_12_1$	0.169
Egg white lysozyme	1.14	31283	1199	$P1$	0.165
RNase A	1.58	20848	1094	$P3_221$	0.212
Laccase	1.2	160473	4638	$P2_12_12_1$	0.213

† Starting values as given by *FMLSQ*.

rithm. Our FFT engine of choice is *FFTW* (Frigo & Johnson, 2005; Johnson & Frigo, 2007).

The excellent *SYMMLQ* routine was included as a part of the *FMLSQ* algorithm for implicit solution of linear systems of equations. *SYMMLQ* is intended for sparse matrices (but performs well with dense) and requires a user-supplied routine for evaluation of matrix–vector products. See Paige & Saunders (1975) for details.

All computation including FFT was performed in double precision since in many cases one has to deal with very ill-conditioned normal matrices (Cowtan & Ten Eyck, 2000). A linear search routine based on function approximation by a cubic polynomial has been used for parameter shift scaling (Press *et al.*, 1992). The appropriate *CCP4* libraries have been used for input/output operations (Collaborative Computational Project, Number 4, 1994).

We have not yet included any geometry terms in the minimized function. The sparse matrix of second derivatives for the geometry terms can be calculated explicitly using conventional methods (Hendrickson & Konnert, 1980). We obtain the following linear system:

$$\mathbf{H}_{\rm total}\Delta\mathbf{p} = \mathbf{H}_{\rm X\text{-}ray}\Delta\mathbf{p} + k\mathbf{H}_{\rm geom}\Delta\mathbf{p} = -(\mathbf{g}_{\rm X\text{-}ray} + k\mathbf{g}_{\rm geom}), \tag{62}$$

where k is a suitably chosen weighting factor. The product $\mathbf{H}_{\rm geom}\Delta\mathbf{p}$ is calculated explicitly using standard linear algebra methods, while the product $\mathbf{H}_{\rm X\text{-}ray}\Delta\mathbf{p}$ is evaluated implicitly using the methods described in this paper. Geometry restraints act as additional observations and are known to stabilize crystallographic refinement (Cowtan & Ten Eyck, 2000) provided that proper weighting schemes are used. Programming of the geometry restraints is in progress.

4. Results and discussion

The *FMLSQ* algorithm has been tested on several small artificial three-atom structures in various space groups. The normal matrix has been calculated using two methods – *via* the conventional (one-plane-all-atoms) approach (Hendrickson & Konnert, 1980) and using the matrix–vector product algorithm described above. In the latter case equation (58) has been used to gain access to matrix elements and analyze the accuracy of the results. It was established that suitable larger integration radii allow one to achieve very accurate results, so that matrices obtained by the conventional approach do not deviate much from matrices obtained by FFT through the matrix–vector product algorithm. If we define $h_{ij}^{\rm conv}$ and $h_{ij}^{\rm FFT}$ as matrix elements calculated using the conventional approach and matrix elements calculated using our FFT algorithm, respectively, then the ratios $h_{ij}^{\rm conv}/h_{ij}^{\rm FFT}$ do not deviate from unity by more than 10^{-7}–10^{-8} in most cases. The tests using equation (58) revealed also that the symmetry of the FFT matrices with respect to the main diagonal was preserved at the levels close to the machine precision.

All calculations have been carried out on a single Intel 2.8 GHz Xeon processor using the Intel Fortran compiler.

4.1. Testing the program on smaller structures

Three structures from the Protein Data Bank (PDB; Berman *et al.*, 2002) and two in-house data sets have been chosen for numerical tests. The PDB structures are α-conotoxin PnIB (PDB code 1akg), hen egg white lysozyme (PDB code 1v7s) and β-galactosidase (PDB codes 1bgl and 1bgm). Structure factors and refined models of ribonuclease A and laccase (Polyakov, 2008) have been kindly provided for these tests by Dr Kostya Polyakov.

Almost all of the refinement runs were carried out to test the correctness of the algorithm and estimate the CPU time for one cycle of refinement. Obviously, unrestrained refinement results should be taken with great caution even at high resolution. For this reason we did not make any attempts to analyze stereochemistry and/or the quality of the resulting models.

For all refinements a five-Gaussian approximation for atomic form factors has been used. During the refinement runs positional and thermal parameters of all atoms have been varied (except for the model of β-galactosidase, where only positional parameters were varied).

Table 1 contains basic crystallographic data for the four structures participating in the numerical tests (β-galactosidase refinement will be discussed below). Five cycles of full-matrix refinement have been carried out for each structure. The detailed CPU time statistics for these runs are highlighted in Table 2. The CPU times are approximately proportional to the size of the structure. The CPU time for the calculation of various preconditioners depends linearly on the number of symmetry operators.

The effectiveness of diagonal preconditioners was tested for the lysozyme structure. When the diagonal preconditioner was removed from the calculations by setting all its elements to unity, the number of iterations increased almost tenfold and varied from 2009 to 5296 (and thus was comparable to the number of refinable parameters). The average CPU time exceeded 4 h per cycle. This shows that inclusion of (at least) a simple diagonal preconditioner is a must for speedy calculations.

A few percent drop in R values was observed for all structures. However, because of the unrestrained nature of these refinements these numbers are not very meaningful (except for the refinement of the small α-conotoxin structure where the R value dropped to 0.148).

Table 2
Full-matrix refinement using a diagonal preconditioner.

Protein	No. of parameters	Iterations per cycle (min–max)	CPU time for one matrix–vector product calculation (s)	Average CPU time per cycle (min)	CPU time for preconditioner calculations (s)
α-Conotoxin PnIB	544	18–21	1.5	1.2	34
Egg white lysozyme	4793	71–103	13.2	23	165
RNase A	4376	54–149	12.1	26	542
Laccase	18552	57–181	56.6	140	2737

4.2. β-Galactosidase as an ultimate CPU time test

As an ultimate test it was decided to run a few cycles of full-matrix *xyz* refinement on the structure of β-galactosidase. β-Galactosidase is a very large protein. It consists of 132 654 non-H atoms. The number of refinable parameters is 397 961. The data set for this protein contained 530 130 unique reflections in space group $P2_1$ at 2.5 Å resolution. Technically, it would be impossible to calculate or get access to all 1.52×10^{11} normal matrix elements on ordinary computers.

One cycle of the refinement on average took 11 d. Five cycles of full-matrix refinement using a diagonal preconditioner required about 56 d of CPU time. Calculation of the diagonal preconditioner took just 2.7 h of CPU time for each cycle of refinement. During five cycles of the refinement, 5121 iterations were made to solve five systems of normal equations. One matrix–vector product evaluation costs approximately 15–16 min in terms of CPU time for this protein structure. Interestingly, if the normal matrix elements were explicitly available the matrix–vector product evaluation (in double precision) would take about the same amount of time ($\sim$13 min).

During the course of the refinement the *R* value dropped from 0.195 to 0.140. The maximal *xyz* shift was about 3.8 Å. This clearly suggests that the normal matrix was ill-conditioned at 2.5 Å resolution and stresses the need for the introduction of tight restraints (Ten Eyck, 1999).

In order to estimate the effect of preconditioning one cycle of refinement was rerun with a block-diagonal preconditioner. One block of the preconditioner included all atoms belonging to the same residue. In total 10 273 327 matrix elements have been calculated using the method described by Tronrud (1999). This calculation consumed about 16 h of CPU time. However, only 469 iterations were required (*versus* 1318 for the first cycle in the previous example) to solve the normal equations. One cycle of the refinement took 140 h (5.8 d), which compares favorably with the CPU time for the first cycle of refinement when a simple diagonal preconditioner was used (15 d of CPU time).

This example clearly shows that a suitable choice of preconditioner may have a rather dramatic effect on the CPU time required to carry out one cycle of full-matrix least-squares refinement. Though the CPU times mentioned above may seem excessive, we should not forget about the size of this structure. We also hope that inclusion of stereochemical restraints with appropriate weights according to equation (62) and other algorithmic improvements will condition the normal matrix and further reduce the number of required iterations to solve the normal equations. Obviously, for full-matrix refinement of such a structure some kind of parallelism is needed.

Memory requirements were high but reasonable for a protein of such a size. Storage of all necessary arrays including three electron-density maps (in double precision) required around 1.1 GB of physical memory.

5. Conclusion

We believe that the matrix–vector product algorithm described above will make full-matrix least-squares refinement feasible for most macromolecular structures on a daily basis owing to its ability to avoid explicit calculation and storage of the normal matrix. CPU time requirements are reasonable for most protein structures. Since nowadays most computers have physical memory of 1 GB or more, handling several electron-density maps in computer memory should not cause difficulties and does not require special equipment.

There is almost no doubt that the approach described above could be extended to calculation of the normal matrix–vector product in the case of anisotropic refinement. However, technically the derivation will be more complicated since 100 sets of coefficients must be considered and summed.

While error analysis for smaller proteins should not cause any difficulties, it is still too soon to judge how useful this method will be for estimating the accuracy of large protein structures (more than, say, 20 000 atoms in the asymmetric part of the unit cell), since calculation of the diagonal elements of the normal matrix inverse remains expensive in terms of CPU time. The simplest and rather obvious recommendation is to calculate s.u. values for a limited set of user-selected atoms or residues on ordinary computers using iterative methods. Better computer equipment (including powerful modern servers with many CPUs) may help to calculate s.u. values for all atoms for large proteins. Inversion of very large matrices is a hot topic of modern applied linear algebra which needs further active research.

APPENDIX *A*
Comment on non-*P*1 space groups

The matrix–vector product expressions can be obtained for any desired space group by application of the chain rule (Urzhumtsev & Lunin, 2001). When applying the chain rule to the matrix–vector product one deals with linear symmetry operators, which allow certain simplifications, and the final result boils down to

$$\begin{pmatrix} [\mathbf{H}\Delta\mathbf{p}]_x^{\rm true} \\ [\mathbf{H}\Delta\mathbf{p}]_y^{\rm true} \\ [\mathbf{H}\Delta\mathbf{p}]_z^{\rm true} \end{pmatrix} = \sum_{m=1}^{N_{\rm sym}} \mathbf{A}_m^{\rm T} \begin{pmatrix} [\mathbf{H}\Delta\mathbf{p}]_{xm}^{P1} \\ [\mathbf{H}\Delta\mathbf{p}]_{ym}^{P1} \\ [\mathbf{H}\Delta\mathbf{p}]_{zm}^{P1} \end{pmatrix}, \tag{63}$$

where $\mathbf{A}_m^{\rm T}$ is the transposed matrix part of the mth symmetry operator for the true space group. This means the matrix–vector product behaves with respect to symmetry like a gradient of the minimized function $f(\mathbf{p})$. Transformation of $[\mathbf{H}\Delta\mathbf{p}]_B$ and $[\mathbf{H}\Delta\mathbf{p}]_O$ is easy, *i.e.* $[\mathbf{H}\Delta\mathbf{p}]_B^{\rm true} = \sum_m [\mathbf{H}\Delta\mathbf{p}]_{Bm}^{P1}$.

In an orthogonal system of coordinates a set of useful identities,

$$[\mathbf{OAD}]^{\rm T} = [\mathbf{OAD}]^{-1} = \mathbf{D}^{-1}\mathbf{A}^{-1}\mathbf{O}^{-1} = \mathbf{O}\mathbf{A}^{-1}\mathbf{D}, \tag{64}$$

might be used to transform the rotational part $\mathbf{A}$ of the symmetry operator. $\mathbf{O}$ is the orthogonalization matrix, $\mathbf{D}$ is the deorthogonalization matrix, $\mathbf{OD} = \mathbf{I}$ and $[\mathbf{OAD}]$ is a unitary three-by-three matrix.

Equation (63) is applicable in both real and reciprocal space after appropriate expansions.

APPENDIX *B*
Other possibilities to derive the matrix–vector product

Let us consider expression (45) again:

$$\sum_{\mathbf{s}} O_i g_i(\mathbf{s}) 4\pi^2 h^2 w(\mathbf{s}) E^x(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}_i). \tag{65}$$

The convolution theorem can be applied to this expression in such a way that

$$\rho_{xx}(\mathbf{r}) = \sum_{\mathbf{s}} w(\mathbf{s}) E^x(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}) \tag{66}$$

may be convoluted with the Fourier transform of $4\pi^2 h^2 O_i g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)$. Using an orthogonal system of coordinates we obtain then

$$\mathcal{F}^{-1}\big[4\pi^2 H_{\rm o}^2 O_i\, g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)\big] = O_i \sum_l^{\rm ngauss} a_l \left(\frac{4\pi}{B_i^{\rm iso} + b_l}\right)^{3/2} \times \left(\frac{4\pi^2}{B_i^{\rm iso} + b_l}\right)\left(2 - \frac{4\pi^2}{B_i^{\rm iso} + b_l} 4\Delta x_i^2\right) \exp\left(-\frac{4\pi^2 r^2}{B_i^{\rm iso} + b_l}\right). \tag{67}$$

$\Delta x_i^2 + \Delta y_i^2 + \Delta z_i^2 = r^2$. Δx_i^2 is the squared distance between the map grid point and the atom center along the x axis. This expression is equivalent to a part of equation (11), which was given by Tronrud (1999) in the compressed form for nine terms at once. Note that in our case the double summation is no longer needed and the metric tensor $\mathbf{G} = \mathbf{I}$ for any orthogonal coordinate frame. One can find all necessary expressions for this approach in the paper mentioned above.

Using this technique and consulting equations (25)–(29), it is clear that the following five maps have to be calculated:

$$\rho_x(\mathbf{r}) = \sum_{\mathbf{s}} w(\mathbf{s}) E^x(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}), \tag{68}$$

$$\rho_y(\mathbf{r}) = \sum_{\mathbf{s}} w(\mathbf{s}) E^y(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}), \tag{69}$$

$$\rho_z(\mathbf{r}) = \sum_{\mathbf{s}} w(\mathbf{s}) E^z(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}), \tag{70}$$

$$\rho_B(\mathbf{r}) = \sum_{\mathbf{s}} w(\mathbf{s}) E^B(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}), \tag{71}$$

$$\rho_O(\mathbf{r}) = \sum_{\mathbf{s}} w(\mathbf{s}) E^O(\mathbf{s}) \exp(-2\pi i \mathbf{s}\mathbf{r}). \tag{72}$$

However, the convolution step becomes cumbersome, *e.g.*

$$\begin{aligned} [\mathbf{H}\Delta\mathbf{p}]_{5(i-1)+1} &= \mathcal{F}^{-1}\big[4\pi^2 H_{\rm o}^2 O_i\, g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)\big] \otimes \rho_x(\mathbf{r}) \\ &+ \mathcal{F}^{-1}\big[4\pi^2 H_{\rm o} K_{\rm o} O_i\, g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)\big] \otimes \rho_y(\mathbf{r}) \\ &+ \mathcal{F}^{-1}\big[4\pi^2 H_{\rm o} L_{\rm o} O_i\, g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)\big] \otimes \rho_z(\mathbf{r}) \\ &+ \mathcal{F}^{-1}\big[\pi i H_{\rm o}(s^2/2) O_i\, g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)\big] \otimes \rho_B(\mathbf{r}) \\ &- \mathcal{F}^{-1}\big[2\pi i H_{\rm o} O_i\, g_i(\mathbf{s}) \exp(2\pi i \mathbf{s}\mathbf{r}_i)\big] \otimes \rho_O(\mathbf{r}). \end{aligned} \tag{73}$$

The expressions for the other rows can be derived in a similar fashion using equations (26)–(29).

Though this approach has been implemented in *FMLSQ*, we found it less numerically accurate than the approach described in the main body of the paper.[8] It is also 1.2–1.3 times slower than the approach defined by equations (48)–(52).

APPENDIX *C*
Bilinear forms and other normal matrix–vector products *via* FFT

The product $\mathbf{u}^{\rm T}\mathbf{H}\mathbf{q}$ can be calculated simply as the inner (dot) product $\langle \mathbf{u}, \mathbf{H}\mathbf{q}\rangle$ since the matrix–vector product algorithm is assumed to be known. However, brief investigation of other possibilities for calculation of the bilinear product $\mathbf{u}^{\rm T}\mathbf{H}\mathbf{q}$ using FFT might be useful. The bilinear form may be rewritten as

$$\begin{aligned} \mathbf{u}^{\rm T}\mathbf{H}\mathbf{q} &= \sum_{i=1}^N u_{5(i-1)+1}[\mathbf{H}\mathbf{q}]_{5(i-1)+1} + \sum_{i=1}^N u_{5(i-1)+2}[\mathbf{H}\mathbf{q}]_{5(i-1)+2} \\ &+ \sum_{i=1}^N u_{5(i-1)+3}[\mathbf{H}\mathbf{q}]_{5(i-1)+3} + \sum_{i=1}^N u_{5(i-1)+4}[\mathbf{H}\mathbf{q}]_{5(i-1)+4} \\ &+ \sum_{i=1}^N u_{5i}[\mathbf{H}\mathbf{q}]_{5i}. \end{aligned} \tag{74}$$

[8] One possible explanation of this observation is the following. The second approach described here is, in fact, equivalent to convolution of second derivatives of atomic densities with the electron-density maps (Murshudov *et al.*, 1997). However, in this case instead of joint atoms with large integration radii we end up with single atom images, and this alone may lead to some loss of precision.

To get an idea of how the final result looks it is enough to consider the first sum in equation (74):

$$\begin{aligned}\mathbf{u}^{\rm T}_{5(i-1)+1}\mathbf{Hq} \\ = &\sum_{\mathbf{s}} 4\pi^2h^2w(\mathbf{s})E^x(\mathbf{s})\sum_{i=1}^{N}u^x_{5(i-1)+1}O_i\,g_i(\mathbf{s})\,\exp(-2\pi i\mathbf{sr}_i) \\ &+\sum_{\mathbf{s}} 4\pi^2hkw(\mathbf{s})E^y(\mathbf{s})\sum_{i=1}^{N}u^x_{5(i-1)+1}O_i\,g_i(\mathbf{s})\,\exp(-2\pi i\mathbf{sr}_i) \\ &+\sum_{\mathbf{s}} 4\pi^2hlw(\mathbf{s})E^z(\mathbf{s})\sum_{i=1}^{N}u^x_{5(i-1)+1}O_i\,g_i(\mathbf{s})\,\exp(-2\pi i\mathbf{sr}_i) \\ &+\sum_{\mathbf{s}} \pi ih(s^2/2)w(\mathbf{s})E^B(\mathbf{s})\sum_{i=1}^{N}u^x_{5(i-1)+1}O_i\,g_i(\mathbf{s})\,\exp(-2\pi i\mathbf{sr}_i) \\ &-\sum_{\mathbf{s}} 2\pi ihw(\mathbf{s})E^O(\mathbf{s})\sum_{i=1}^{N}u^x_{5(i-1)+1}O_i\,g_i(\mathbf{s})\exp(-2\pi i\mathbf{sr}_i).\end{aligned} \tag{75}$$

Dividing now vector $\mathbf{u}^{\rm mod}$ into the following five components,

$$\mathbf{u}^{\rm mod} = \begin{cases} u^x_{5(i-1)+1}O_i \\ u^y_{5(i-1)+2}O_i \\ u^z_{5(i-1)+3}O_i & i = 1, 2, \ldots, N \\ u^B_{5(i-1)+4}O_i \\ u^O_{5i} \end{cases} \tag{76}$$

and setting

$$\sum_{i=1}^{N}u^x_{5(i-1)+1}O_ig_i(\mathbf{s})\exp(-2\pi i\mathbf{sr}_i) = F^{ux*}_{\rm c}(\mathbf{s}), \tag{77}$$

$$\sum_{i=1}^{N}u^y_{5(i-1)+2}O_ig_i(\mathbf{s})\exp(-2\pi i\mathbf{sr}_i) = F^{uy*}_{\rm c}(\mathbf{s}), \tag{78}$$

$$\sum_{i=1}^{N}u^z_{5(i-1)+3}O_ig_i(\mathbf{s})\exp(-2\pi i\mathbf{sr}_i) = F^{uz*}_{\rm c}(\mathbf{s}), \tag{79}$$

$$\sum_{i=1}^{N}u^B_{5(i-1)+4}O_ig_i(\mathbf{s})\exp(-2\pi i\mathbf{sr}_i) = F^{uB*}_{\rm c}(\mathbf{s}), \tag{80}$$

$$\sum_{i=1}^{N}u^O_{5i}g_i(\mathbf{s})\exp(-2\pi i\mathbf{sr}_i) = F^{uO*}_{\rm c}(\mathbf{s}), \tag{81}$$

we are ready to write out the final result for the $P1$ space group:

$$\begin{aligned}\mathbf{u}^{\rm T}\mathbf{Hq} = &\sum_{\mathbf{s}} 4\pi^2h^2w(\mathbf{s})F^{ux*}_{\rm c}(\mathbf{s})E^x(\mathbf{s}) + \sum_{\mathbf{s}} 4\pi^2hkw(\mathbf{s})F^{ux*}_{\rm c}(\mathbf{s})E^y(\mathbf{s}) \\ &+\sum_{\mathbf{s}} 4\pi^2hlw(\mathbf{s})F^{ux*}_{\rm c}(\mathbf{s})E^z(\mathbf{s}) + \sum_{\mathbf{s}} \pi ih(s^2/2)w(\mathbf{s})F^{ux*}_{\rm c}(\mathbf{s})E^B(\mathbf{s}) \\ &-\sum_{\mathbf{s}} 2\pi ihw(\mathbf{s})F^{ux*}_{\rm c}(\mathbf{s})E^O(\mathbf{s}) + \sum_{\mathbf{s}} 4\pi^2hkw(\mathbf{s})F^{uy*}_{\rm c}(\mathbf{s})E^x(\mathbf{s}) \\ &+\sum_{\mathbf{s}} 4\pi^2k^2w(\mathbf{s})F^{uy*}_{\rm c}(\mathbf{s})E^y(\mathbf{s}) + \sum_{\mathbf{s}} 4\pi^2klw(\mathbf{s})F^{uy*}_{\rm c}(\mathbf{s})E^z(\mathbf{s}) \\ &+\sum_{\mathbf{s}} \pi ik(s^2/2)w(\mathbf{s})F^{uy*}_{\rm c}(\mathbf{s})E^B(\mathbf{s}) - \sum_{\mathbf{s}} 2\pi ikw(\mathbf{s})F^{uy*}_{\rm c}(\mathbf{s})E^O(\mathbf{s}) \\ &+\sum_{\mathbf{s}} 4\pi^2hlw(\mathbf{s})F^{uz*}_{\rm c}(\mathbf{s})E^x(\mathbf{s}) + \sum_{\mathbf{s}} 4\pi^2klw(\mathbf{s})F^{uz*}_{\rm c}(\mathbf{s})E^y(\mathbf{s}) \\ &+\sum_{\mathbf{s}} 4\pi^2l^2w(\mathbf{s})F^{uz*}_{\rm c}(\mathbf{s})E^z(\mathbf{s}) + \sum_{\mathbf{s}} \pi il(s^2/2)w(\mathbf{s})F^{uz*}_{\rm c}(\mathbf{s})E^B(\mathbf{s}) \\ &-\sum_{\mathbf{s}} 2\pi ilw(\mathbf{s})F^{uz*}_{\rm c}(\mathbf{s})E^O(\mathbf{s}) - \sum_{\mathbf{s}} \pi ih(s^2/2)w(\mathbf{s})F^{uB*}_{\rm c}(\mathbf{s})E^x(\mathbf{s}) \\ &-\sum_{\mathbf{s}} \pi ik(s^2/2)w(\mathbf{s})F^{uB*}_{\rm c}(\mathbf{s})E^y(\mathbf{s}) \\ &-\sum_{\mathbf{s}} \pi il(s^2/2)w(\mathbf{s})F^{uB*}_{\rm c}(\mathbf{s})E^z(\mathbf{s}) + \sum_{\mathbf{s}} (s^4/16)w(\mathbf{s})F^{uB*}_{\rm c}(\mathbf{s})E^B(\mathbf{s}) \\ &-\sum_{\mathbf{s}} (s^2/4)w(\mathbf{s})F^{uB*}_{\rm c}(\mathbf{s})E^O(\mathbf{s}) + \sum_{\mathbf{s}} 2\pi ihw(\mathbf{s})F^{uO*}_{\rm c}(\mathbf{s})E^x(\mathbf{s}) \\ &+\sum_{\mathbf{s}} 2\pi ikw(\mathbf{s})F^{uO*}_{\rm c}(\mathbf{s})E^y(\mathbf{s}) + \sum_{\mathbf{s}} 2\pi ilw(\mathbf{s})F^{uO*}_{\rm c}(\mathbf{s})E^z(\mathbf{s}) \\ &-\sum_{\mathbf{s}} (s^2/4)w(\mathbf{s})F^{uO*}_{\rm c}(\mathbf{s})E^B(\mathbf{s}) + \sum_{\mathbf{s}} w(\mathbf{s})F^{uO*}_{\rm c}(\mathbf{s})E^O(\mathbf{s}).\end{aligned} \tag{82}$$

Thus after generation of appropriate $F_{\rm c}$ values the whole calculation can be carried out in reciprocal space. Convolution in real space is not really needed. Clearly, on the basis of equation (82) we can calculate the quadratic form $\mathbf{q}^{\rm T}\mathbf{Hq}$ which might be required for certain applications.

Using the matrix–vector product alone it is also easy to calculate various other quantities: $\mathbf{H}^2\mathbf{q} = \mathbf{H}(\mathbf{Hq})$, $\mathbf{q}^{\rm T}\mathbf{Hq} = \langle\mathbf{q}, \mathbf{Hq}\rangle$, $\mathbf{q}^{\rm T}\mathbf{H}^2\mathbf{q} = \langle\mathbf{q}, \mathbf{H}(\mathbf{Hq})\rangle$ *etc.* However, quadratic forms like $\mathbf{q}^{\rm T}\mathbf{H}^{-1}\mathbf{q}$ may require solution of the linear system $\mathbf{Hx} = \mathbf{q}$.

APPENDIX *D*
Preconditioners

Solution of very large linear systems of equations is not easy. The rate of convergence of the iterative method depends on the eigenvalue spectrum of the $5N$-by-$5N$ $\mathbf{H}$ matrix. Without preconditioning it usually takes about $n = 5N$ iterations to solve the system of equations

$$\mathbf{Hx} = \mathbf{g}. \tag{83}$$

Therefore, any iterative method usually involves another matrix $\mathbf{Q}$, which transforms matrix $\mathbf{H}$ into one with a more favorable spectrum. This transformation matrix $\mathbf{Q}$ is known as the 'preconditioner'.

In general, several preconditioning schemes are available:

$$\mathbf{Q}^{-1}\mathbf{Hx} = \mathbf{Q}^{-1}\mathbf{g} \quad \text{(left preconditioning)} \tag{84}$$

or

$$(\mathbf{HQ}^{-1})\mathbf{Qx} = \mathbf{g} \quad \text{(right preconditioning)} \tag{85}$$

or

$$\mathbf{Q}_{\rm L}^{-1}\mathbf{H}\mathbf{Q}_{\rm R}^{-1}(\mathbf{Q}_{\rm R}\mathbf{x}) = \mathbf{Q}_{\rm L}^{-1}\mathbf{g} \quad \text{(two-sided preconditioning),} \tag{86}$$

where $\mathbf{Q} = \mathbf{Q}_{\rm L}\mathbf{Q}_{\rm R}$. To be able to reduce the number of iterations, one must try to get $\mathbf{Q}$ as close as possible to $\mathbf{H}$. In this case eigenvalues of $\mathbf{Q}^{-1}\mathbf{H}$ are closer to 1 in magnitude and thus any iterative method using this matrix as a basis for its iterative step will have much better convergence. The choice of preconditioning scheme depends on the particular algorithm.

Diagonal preconditioners and to some extent general preconditioners have been reviewed in some detail by Cowtan & Ten Eyck (2000) and we refer the reader to this useful paper.

Since diagonal and arbitrary off-diagonal matrix elements are not easily accessible, the method developed by Tronrud (1999) is actively used in the *FMLSQ* program for calculation of various preconditioners. Note that calculation of a good (*e.g.* block-diagonal) preconditioner may require significant CPU time. The major goal is to find such a preconditioner that the overall time for one cycle of refinement is close to minimal.

We are indebted to Dr Kostya Polyakov and Dr Tatyana Safonova for carefully reading the manuscript and useful suggestions. Special thanks are due to the group developing the *LyX* editor (http://www.lyx.org), whose work greatly simplified the writing and processing of this article. This work was supported by the Russian Federation State grant No. 02.513.11.3388 within the program 'Research and development along the priority directions for scientifico-technological complex of Russia in 2007–2012'.

References

Agarwal, R. C. (1978). *Acta Cryst.* A**34**, 791–809.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. & Sorensen, D. (1999). *LAPACK, Users' Guide*, 3rd ed. Philadelphia: Society for Industrial and Applied Mathematics.

Berman, H. M. *et al.* (2002). *Acta Cryst.* D**58**, 899–907.

Brünger, A. T., Adams, P. D., Clore, G. M., DeLano, W. L., Gros, P., Grosse-Kunstleve, R. W., Jiang, J.-S., Kuszewski, J., Nilges, M., Pannu, N. S., Read, R. J., Rice, L. M., Simonson, T. & Warren, G. L. (1998). *Acta Cryst.* D**54**, 905–921.

Collaborative Computational Project, Number 4 (1994). *Acta Cryst.* D**50**, 760–763.

Cowtan, K. & Ten Eyck, L. F. (2000). *Acta Cryst.* D**56**, 842–856.

Frigo, M. & Johnson, S. G. (2005). *Proc. IEEE*, **93**, 216–231.

Hendrickson, W. A. & Konnert, J. H. (1980). *Computing in Crystallography*, edited by R. Diamond, S. Ramseshan & K. Venkatesan, pp. 13.01–13.26. Bangalore: Indian Academy of Sciences.

Hestenes, M. R. & Stiefel, E. (1952). *J. Res. Natl Bur. Stand.* **49**, 409–436.

Johnson, S. G. & Frigo, M. (2007). *IEEE Trans. Signal Proc.* **55**, 111–119.

Kim, K. M., Nesterov, Yu. E. & Cherkassky, B. V. (1984). *Dokl. Acad. Nauk SSSR*, **275**, 1306–1309.

Lanczos, C. (1952). *J. Res. Natl Bur. Stand.* **49**, 33–53.

Lifchitz, A. (1986). *Acta Cryst.* A**42**, 204.

Lunin, V. Yu. & Urzhumtsev, A. G. (1985). *Acta Cryst.* A**41**, 327–333.

Murshudov, G. N., Vagin, A. A. & Dodson, E. J. (1997). *Acta Cryst.* D**53**, 240–255.

Navaza, J. (2002). *Acta Cryst.* A**58**, 568–573.

Paige, C. C. & Saunders, M. A. (1975). *SIAM J. Numer. Anal.* **12**, 617–629.

Polyakov, K. M. (2008). Unpublished work.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T. (1992). *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing*, 2nd ed. Cambridge University Press.

Sheldrick, G. M. (2008). *Acta Cryst.* A**64**, 112–122.

Ten Eyck, L. F. (1999). *Crystallographic Computing 7*, edited by P. E. Bourne & K. Watenpaugh. Oxford University Press.

Ten Eyck, L. F. (2003). *Methods Enzymol.* **374**, 345–369.

Tronrud, D. E. (1997). *Methods Enzymol. B*, **277**, 306–319.

Tronrud, D. E. (1999). *Acta Cryst.* A**55**, 700–703.

Urzhumtsev, A. G. & Lunin, V. Y. (2001). *Acta Cryst.* A**57**, 451–460.