TomoEED: Fast Edge-Enhancing Denoising of Tomographic Volumes – Supplementary Information –

J.J. Moreno, A. Martínez-Sánchez, J.A. Martínez, E.M. Garzón, J.J. Fernández

S1. Optimized implementation of 3D Anisotropic Nonlinear Diffusion in TomoEED

S1.1. Numerical discretization of the diffusion equation

AND follows the diffusion equation

$$I_t = \mathrm{div}(\mathbf{D} \cdot \nabla I) \tag{S1}$$

where I_t denotes the derivative with respect to the time, $\nabla I = (I_x, I_y, I_z)$ is the gradient vector of the volume I and div is the divergence operator. The 3×3 matrix $\mathbf{D}$ is the diffusion tensor and tunes the filtering according to the local structure, as described in the main text.

This diffusion equation (Eq. S1) can be numerically discretized using finite differences. The term $I_t = \frac{\partial I}{\partial t}$ can be replaced by an Euler forward difference approximation. The resulting explicit scheme allows calculation of subsequent versions of the volume iteratively:

$$\begin{aligned} I^{(k)} = I^{(k-1)} + \tau \cdot (\quad & \tfrac{\partial}{\partial x}(D_{11} I_x) + \tfrac{\partial}{\partial x}(D_{12} I_y) + \tfrac{\partial}{\partial x}(D_{13} I_z) + \\ & \tfrac{\partial}{\partial y}(D_{21} I_x) + \tfrac{\partial}{\partial y}(D_{22} I_y) + \tfrac{\partial}{\partial y}(D_{23} I_z) + \\ & \tfrac{\partial}{\partial z}(D_{31} I_x) + \tfrac{\partial}{\partial z}(D_{32} I_y) + \tfrac{\partial}{\partial z}(D_{33} I_z)) \end{aligned} \tag{S2}$$

where τ denotes the time step size, $I^{(k)}$ denotes the volume at time $t_k = k\tau$ and the D_{mn} terms represent the components of the symmetric diffusion tensor $\mathbf{D}$, with $D_{mn} = D_{nm}$ because of the symmetry. The spatial derivatives ($\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$ and $\frac{\partial}{\partial z}$) are approximated based on central differences. For the sake of numerical stability, the maximum time step size in this case is $\tau = 0.1$.

S1.2. High level algorithm of Anisotropic Nonlinear Diffusion

The algorithm in AND can be expressed as:

---

1. Compute the structure tensor $\mathbf{J}$.
For all voxels, the structure tensor is computed as described in the main text (Eq. 1).

2. Compute the diffusion tensor $\mathbf{D}$.
For all voxels, the diffusion tensor is computed by the following steps:

2.1 Eigen-decomposition of the structure tensor $\mathbf{J}$.
This step obtains the eigenvectors $\mathbf{v_i}$ and eigenvalues μ_i, with $\mu_1 \geq \mu_2 \geq \mu_3$. Thus, the first eigenvector $\mathbf{v_1}$ represents the direction of the maximum density variation whereas $\mathbf{v_2}$ and $\mathbf{v_3}$ are directions with lower density change.

2.2 Composition of the diffusion tensor $\mathbf{D}$.
$\mathbf{D}$ is built as described in the main text (Eq. 2), using the eigenvectors $\mathbf{v_i}$ of the structure tensor $\mathbf{J}$, and the filtering along these directions is set by means of the eigenvalues λ_i:
$\lambda_1 = 1.0 - \exp(-3.31488/(|\nabla I|/K)^8)$
λ_2=λ_3=1

3. Solve the partial differential equation of diffusion, Eq. S2.

4. Iterate: go to step 1.

S1.3. Fast eigen-decomposition of symmetric 3×3 matrices in TomoEED

The eigen-decomposition –also often referred to as numerical diagonalization– of a real, symmetric matrix $\mathbf{A}$ consists in calculating a set of eigenvalues λ_i and eigenvectors $\mathbf{v_i}$ satisfying $\mathbf{A}\mathbf{v_i} = \lambda_i \mathbf{v_i}$ that derives in the matrix factorization:

$$\mathbf{A} = \mathbf{V}\mathbf{L}\mathbf{V}^T = [\mathbf{v_1}\,\mathbf{v_2}\,\mathbf{v_3}] \cdot \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \cdot [\mathbf{v_1}\,\mathbf{v_2}\,\mathbf{v_3}]^T \tag{S3}$$

The Jacobi algorithm

The most accurate method for real symmetric matrix diagonalization is the Jacobi algorithm (Press *et al.*, 2002). It essentially consists in applying a sequence of orthogonal similarity transformations of the form $\mathbf{A} \rightarrow \mathbf{P}_j\ \mathbf{A}\ \mathbf{P}_j^T$, aiming to progressively zero the off-diagonal elements (or, rather, making them insignificant) and eventually turning $\mathbf{A}$ into a diagonal matrix to machine precision. The accumulated product of the transformations, $\prod_j \mathbf{P}_j$, gives the matrix of eigenvectors while the elements of the final diagonal matrix are the eigenvalues. Typical matrices require a number of transformations of 3 to 5 times the number of components of the matrix (i.e. a total of 27 to 45 for 3×3 matrices) to achieve convergence, each involving an order of 12×3 operations (Press *et al.*, 2002; Kopp, 2008).

Direct analytical calculation

The quickest way to determine the eigenvalues of a real symmetric 3×3 matrix $\mathbf{A}$ is by directly solving the characteristic equation, $P(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I}) = 0$, where $\mathbf{I}$ is the identity matrix and $\det$ denotes the determinant of the matrix. This characteristic equation can be expressed as a cubic equation:

$$P(\lambda) = \lambda^3 + c_2\lambda^2 + c_1\lambda + c_0 = 0, \text{with} \begin{cases} c_2 = -a_{11} - a_{22} - a_{33} \\ c_1 = a_{11}a_{22} + a_{11}a_{33} + a_{22}a_{33} - a_{12}^2 - a_{13}^2 - a_{23}^2 \\ c_0 = a_{11}a_{23}^2 + a_{22}a_{13}^2 + a_{33}a_{12}^2 - a_{11}a_{22}a_{33} - 2a_{13}a_{12}a_{23} \end{cases} \tag{S4}$$

where a_{ij} are the coefficients of the real symmetric 3×3 matrix $\mathbf{A}$.

Kopp (2008) demonstrated that this equation can be solved by following Cardano's method for the solution of the general cubic equation, and obtained the analytical expression for the three eigenvalues:

$$\lambda_1 = -\tfrac{1}{3}(c_2 + c) + c \qquad \lambda_2 = -\tfrac{1}{3}(c_2 + c) - s \qquad \lambda_3 = -\tfrac{1}{3}(c_2 + c) + s \tag{S5}$$

with $c = \sqrt{p}\cos\phi$, $s = \frac{1}{\sqrt{3}}\sqrt{p}\sin\phi$, $p = c_2^2 - 3c_1$, $\phi = \frac{1}{3}\arctan\frac{\sqrt{27\left[\frac{1}{4}c_1^2(p-c_1) + c_0(q+\frac{27}{4}c_0)\right]}}{q}$
and $q = -c_2(p - \frac{3}{2}c_1) - \frac{27}{2}c_0$.

Next, the eigenvectors can be then efficiently calculated by vector cross products based on the fact that $(\mathbf{A} - \lambda_i\mathbf{I})\mathbf{v}_i = 0$, which leads to $\mathbf{v}_i^T(\mathbf{A} - \lambda_i\mathbf{I})\mathbf{x} = 0$, for any arbitrary vector $\mathbf{x}$, in particular for the unit vectors $\mathbf{e}_1 = (1,0,0)^T$ and $\mathbf{e}_2 = (0,1,0)^T$. Consequently, owing to this orthogonality, $\mathbf{v}_i$ can be calculated as:

$$\mathbf{v}_1 = (\mathbf{A}\mathbf{e}_1 - \lambda_1\mathbf{e}_1) \times (\mathbf{A}\mathbf{e}_2 - \lambda_1\mathbf{e}_2) \qquad \mathbf{v}_2 = (\mathbf{A}\mathbf{e}_1 - \lambda_2\mathbf{e}_1) \times (\mathbf{A}\mathbf{e}_2 - \lambda_2\mathbf{e}_2) \qquad \mathbf{v}_3 = \mathbf{v}_1 \times \mathbf{v}_2 \tag{S6}$$

Kopp's analytical method is very fast, with the most expensive operations being the three trigonometric functions ($\sin$, $\cos$, $\arctan$) in Eq. (S5). However, this method is prone to limited numerical accuracy, though this drawback does not pose a major problem for practical applications that require moderate accuracy (Kopp, 2008). Kopp proposed a hybrid approach where analytical calculation is used by default and, when the procedure is estimated to be inaccurate based on the magnitude of the eigenvalues/eigenvectors, it switches to more precise, conventional calculation. In TomoEED we use Kopp's pure analytical method sketched in this section since we have observed negligible differences for this denoising application (Sections S2.3 and S2.4).

S1.4. Memory-efficient multithreaded implementation of TomoEED.

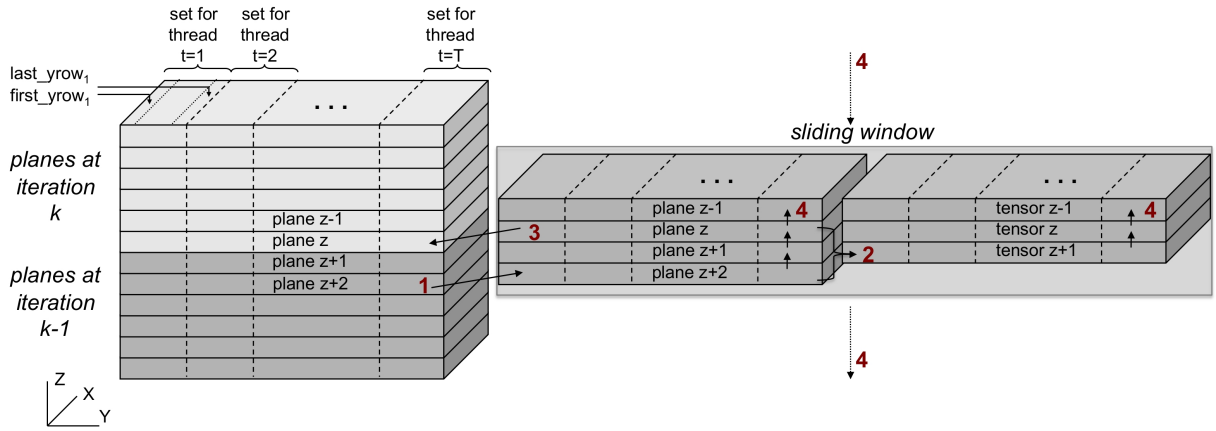

Figure S1: **Memory-efficient multithreaded implementation of TomoEED.**
Only one copy of the volume is kept in memory and it is gradually updated by Z-planes during the iterative process. The sliding window keeps the data needed for the processing of the plane z for the current solution I^k. The arrows and red numbers show the information transfer during the processing of the plane z: (1) the plane z+2 from the previous solution I^{k-1} is got from the volume and stored in the window, (2) the tensors for plane z+1 are then calculated from planes z, z+1 and z+2 available in the window (Steps 1 and 2 of the algorithm in Section S1.2), (3) the processing of the plane z (Step 3 of the algorithm in Section S1.2) is carried out using only the data in the sliding window: planes and tensors from z-1, z and z+1. The resulting plane z is updated and stored in the volume, (4) The planes in the sliding window are pushed backward to make space for a new plane coming from the volume. The dotted lines show how the sliding window is pushed forward for the processing of a new plane.
The multithreaded implementation distributes the volume into sets of Y rows to be processed by the threads in parallel. t denotes the index of the thread. $\mathrm{first\_yrow_t}$ and $\mathrm{last\_yrow_t}$ are the indices of the first and last Y rows of the set assigned to the thread t. For simplicity, only $\mathrm{first\_yrow_1}$ and $\mathrm{last\_yrow_1}$ are shown in the figure.

Distribute the volume into sets of Y rows:
  Compute $\mathrm{first\_yrow_t}$, $\mathrm{last\_yrow_t}$, $\forall t$
For $k = 1 \ldots \mathrm{N_{iters}}$
  Synchronize the threads.
  Initialize the sliding window with the first planes and tensors
  For $z = 1 \ldots \mathrm{N_z}$
    Synchronize the threads.
    Slide the sliding window and include $I^{k-1}(z+2)$.
    Synchronize the threads.
    In parallel, the threads process their sets:
      For $y = \mathrm{first\_yrow_t} \ldots \mathrm{last\_yrow_t}$
        Compute the structure tensors at z+1 in the sliding window.
        Compute the diffusion tensors at z+1 in the sliding window.
        Synchronize the threads.
        Solve the discretized diffusion equation Eq. (S2) for $I^k(z, y, x)$, $\forall x$
          using the planes and tensors at z-1, z and z+1 available in the sliding window.

Figure S2: Algorithm for the memory-efficient multithreaded algorithm in TomoEED. t denotes the index of the thread. $\mathrm{first\_yrow_t}$ and $\mathrm{last\_yrow_t}$ are the indices of the first and last Y-rows of the set assigned to the thread t.

S2. Illustrative results

S2.1. Denoising datasets from different volumetric electron microscopy disciplines

To illustrate the performance of TomoEED and show its potential, we have applied it to datasets from different volume electron microscopy disciplines where feature-preserving noise reduction is needed. Fig. **S3** shows the results on volumes obtained from cryo-ET of Vaccinia virus (Cyrklaff *et al.* PNAS 102:2772–2777, 2005), ET and FIB-SEM of mouse brain tissue plastic sections (Fernandez-Fernandez *et al.*, J. Cell Sci. 130:83-89, 2017). They were obtained with automated, time-varying tuning of the parameter K based on the average gradient of the whole volume. Significant noise reduction and substantial preservation of the main structural biological features are clearly observed. Further demonstration of the utility of the automated parameter tuning is in Section S2.4.

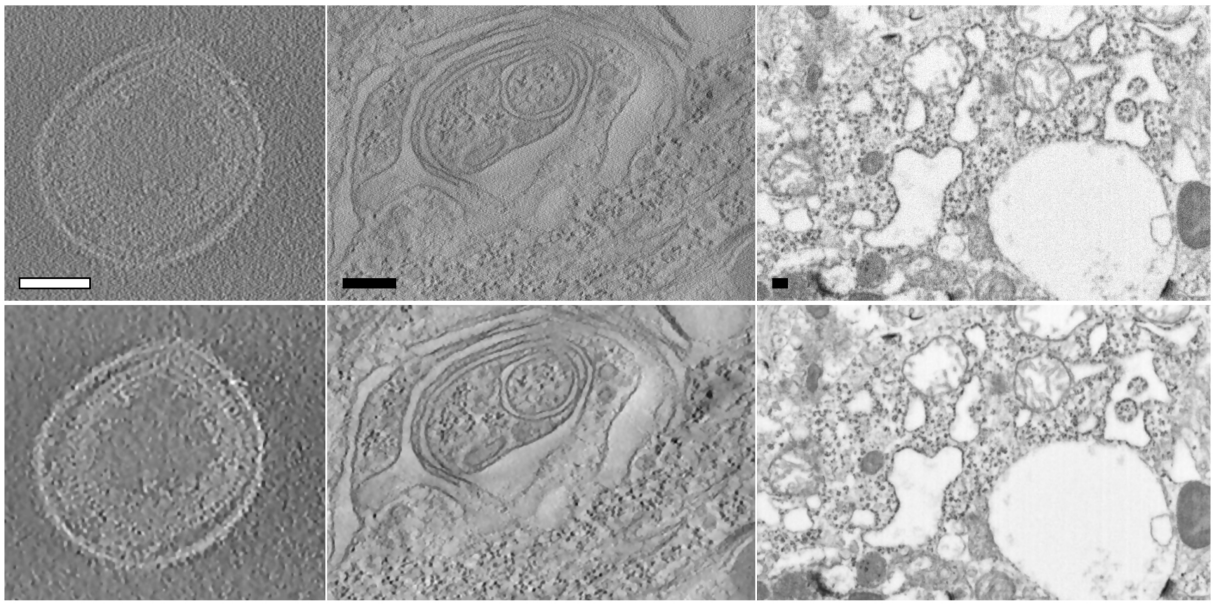

Figure S3: **Noise reduction with TomoEED**. Cryo-ET of Vaccinia virus, ET of HPF/FS (high-pressure freezing/freeze-substitution) mouse brain tissue and FIB-SEM (Focused Ion Beam - Scanning Electron Microscopy) of chemically-fixed mouse brain tissue. Note the good preservation of membranes in all cases, and spikes in the core of the virus (left), ribosomes (centre) and ribosome rosettes and mitochondria cristae (right) whereas the noise has been substantially reduced. White bar: 100nm. Black bars: 200nm.

S2.2. Processing times, speedup factors and memory consumption

Table S1: Processing time (s) in TomoEED

dataset size	Jacobi (1T)	Analytic (1T)	Jacobi (16T)	Analytic (16T)	Jacobi (GPU)*	Analytic (GPU)*
256	60.50	33.96	5.84	3.78	2.80	1.83
384	205.00	113.89	17.62	10.77	8.02	4.72
460	356.33	199.00	29.99	18.77	13.53	7.65
512	498.46	275.61	38.77	24.18	17.79	9.79
640	961.34	530.20	74.73	45.23	34.00	19.11

Table S2: Speedup factors with regard to their sequential (1T) version

dataset size	Jacobi (16T)	Analytic (16T)	Jacobi (GPU)	Analytic (GPU)
256	10.35	8.97	21.62	18.55
384	11.64	10.58	25.56	24.09
460	11.88	10.60	26.34	26.01
512	12.86	11.39	28.01	28.14
640	12.86	11.72	28.27	27.74

Table S3: Comparison with an implementation of AND in a standard package: IMOD
Processing time (s) and memory consumption (GB)

	TomoEED			**IMOD**	
dataset size	Jacobi (1T)	Analytic (1T)	Memory consum.	Jacobi (1T)	Memory consum.
256	60.50	33.96	0.07	81.07	0.64
384	205.00	113.89	0.23	272.92	2.22
460	356.33	199.00	0.39	433.26	3.75
512	498.46	275.61	0.53	648.79	4.63
640	961.34	530.20	1.02	1261.44	8.96

* These GPU results were obtained on a NVIDIA GPU Tesla K80.
TomoEED works on GPUs with Compute Capability 2.0 (Fermi microarchitecture) onwards and the actual performance will depend of the particular specifications of the card.

S2.3. Influence of the limited accuracy matrix diagonalization in AND

Analytical matrix diagonalization has the drawback of limited numerical accuracy. To evaluate the practical influence of this limitation on this denoising application, we computed the relative error between the denoised solutions obtained with the Jacobi algorithm and the analytical strategy for different datasets. The Jacobi solution is considered the most accurate one. This relative error is mathematically formulated as:

$$\frac{\sum_{i=1}^{N} |I_i^j - I_i^a|}{\sum_{i=1}^{N} I_i^j} \tag{S7}$$

where N denotes the number of voxels of the volume, and I_i^j and I_i^a represent the voxels from the denoised solution obtained with the Jacobi algorithm and analytical numerical diagonalization in TomoEED, respectively.

The relative error in Eq. S7 is a global, average measure that might conceal discrete voxels where the Jacobi-based and the analytical solutions could show large differences. In order to unveil these potential situations, we also computed the maximum relative error in individual voxels, relative to the mean density value of the tomogram:

$$\max_{i=1\ldots N} \{ \frac{|I_i^j - I_i^a|}{\frac{1}{N}\sum_{i=1}^{N} I_i^j} \} \tag{S8}$$

Table **S4** shows the relative error and the maximum individual relative error obtained for 5 cryoET test datasets of different sizes (256, 384, 460, 512, 640) used throughout this work, and for 2 datasets of mouse brain tissue plastic sections (ps1, ps2). It is clearly seen that the global relative error is negligible and that the individual relative error in the voxels is kept below 1%.

Table S4: Relative error between denoised volumes using Jacobi-based and analytical matrix diagonalization in TomoEED

Dataset	Relative error	Maximum individual relative error
256	0.00000655	0.00718860
384	0.00000757	0.00754244
460	0.00000756	0.00363749
512	0.00000779	0.00780323
640	0.00000768	0.00585213
ps1	0.00000679	0.00281266
ps2	0.00000658	0.00251588

S2.4. Utility of automated parameter tuning

To illustrate the mechanism in TomoEED to tune the main parameter in AND, K, we focused on a HIV-1 virions tomogram (Briggs *et al.*, Structure 14:15-20, 2006) available in EMDatabank (emd-1155). The tomogram density was rescaled to zero mean and standard deviation of 1. Fig. **S4** shows the results with 10 iterations of AND with TomoEED under different conditions. The result with automated time-varying K is shown in Fig. **S4**(b). Here, K was set in the range from 0.84 to 0.30 based on the average gradient in an area containing only noise (similar results were obtained using the average gradient in the whole tomogram). The denoised result shows reasonable noise reduction, background flattening and preservation of features.

To show the effect of excessively high values of K, we run TomoEED with K=0.84 fixed during the 10 iterations. Here, high gradient is needed to consider voxels as part of edges to preserve, otherwise strong smoothing is applied. As a consequence, the denoised result (Fig. **S4**(c)) turned out to be rather blurred. Higher values of K would produce even more blurred solutions. As an example of a value of K that might be low, TomoEED was run with K=0.30. Here, more voxels are preserved and, accordingly, the effect of the smoothing (Fig. **S4**(d)) is subtler than in (b). Lower values of K would translate into less filtered volumes. Finally, an intermediate value of K (0.57) resulted into a solution (Fig. **S4**(e)) similar to that by automated time-varying (b).

These results show that the automated parameter tuning in TomoEED helps the user find a fairly good denoised solution or, at least, to obtain a range of values for K over which subsequent manual refinement can be conducted.

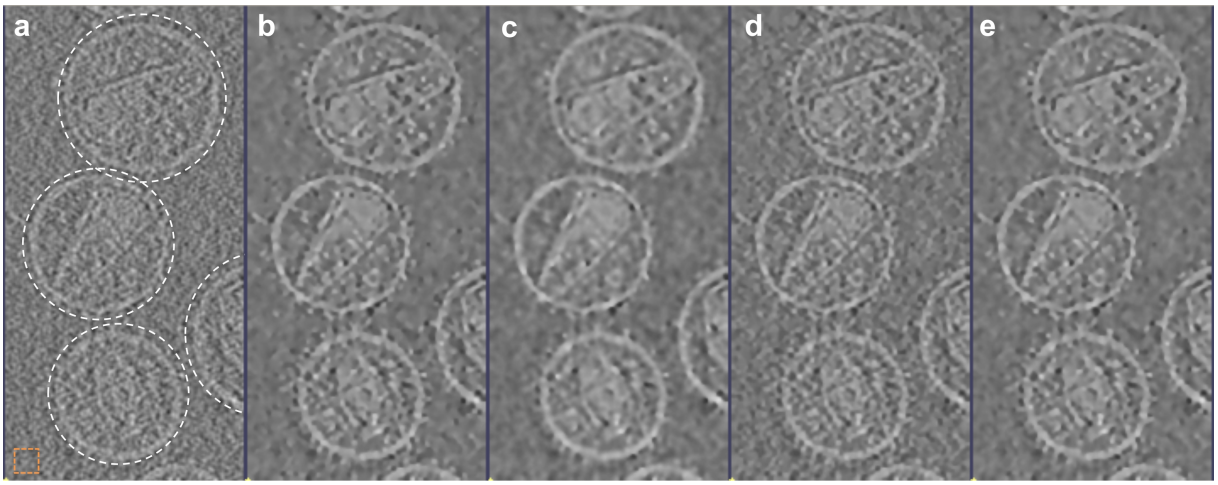

Figure S4: **Comparison of denoising results.** (a) original tomogram, (b) denoised with automated time-varying K tuning, which varied from K=0.84 to 0.30 through 10 iterations of AND, (c) denoised with K fixed to K=0.84 during 10 iterations, (d) denoised with K fixed to K=0.30 during 10 iterations, (e) denoised with K fixed to 0.57 (mean value of 0.84 and 0.30). The automated tuning of K was based on the average gradient measured in a 10x10x10-voxel area containing only noise (highlighted in orange in (a)). Dashed areas in (a) delimite regions that were considered structures of interest (foreground) for calculation of SNR and sharpness measures.

Further assessment was performed based on SNR (signal-to-noise ratio) and sharpness measures. The SNR was estimated as the ratio of the average power in the foreground (structures of interest, see Fig. **S4**(a)) and that in the background. Similarly, sharpness was estimated from the ratio of the average gradient in those regions. Table **S5** shows the results obtained for the tomograms in Fig. **S4**. The metrics were computed from tomograms denoised with TomoEED using the two methods for matrix diagonalization.

Table S5: SNR and sharpness of tomograms in Fig. **S4**(a-e)

	a (original)	b	c	d	e	diagonalization
SNR:	1.70995	3.39708	3.04733	2.48948	3.41752	Analytic
		3.39790	3.04777	2.48882	3.41746	Jacobi
Sharpness:	1.18589	2.08142	1.91055	1.51407	2.04571	Analytic
		2.08176	1.91059	1.51371	2.04558	Jacobi

These metrics quantitatively reflect what is observed in Fig. **S4**. Automated time-varying parameter tuning may provide an acceptable denoised solution or a range of values of K to conduct subsequent refinement. Values beyond that range may translate into poorer solutions because of either blurring or insufficient filtering. Moreover, these results confirm that the difference between the matrix diagonalization methods is negligible from the practical point of view.