A modular software framework for the design and implementation of ptychography algorithms: Supplementary Material

Francesco Guzzi[1, 2], George Kourousias[1], Fulvio Billè[1], Roberto Pugliese[1], Alessandra Gianoncelli[1], and Sergio Carrato[2]

[1]Elettra Sincrotrone Trieste, Basovizza, Italy
[2]University of Trieste, Trieste, Italy
[*]francesco.guzzi@elettra.eu

ABSTRACT

In the following text we will expand the results and discussion section of the main text, providing addition reconstructions for different datasets (optical, soft X-ray and synthetic). We also describe in some detail the pre-processing and post-processing required for some reconstructions. We also describe the behaviour of the position refinement technique while using different error signals.

SciComPty simulator

Reconstructions apart, SciComPty offers a simple method to perform virtual experiments. To simulate a ptychography dataset, one can implement the general transmission model of Eq. 3 in the main text, and assign particular values to its geometrical/setup parameters. In the case of a far-field setup[1], by fixing the detector pixel size δ_d, the corresponding pixel size at the sample plane δ_s is given by:

$$\delta_s = \frac{\lambda \cdot z_{do}}{W \cdot \delta_d} = \frac{\lambda \cdot z_{do}}{S_{detector}} \tag{1}$$

where W is the detector size in pixels, S_d is the detector lateral dimension and z_{do} the sample-detector distance. δ_s defines the basis for the scanning movements, that in the simplest case, follows a grid pattern. Random jitter is added to the (x,y) coordinates, to prevent the raster scanning pathology[2–4]. The overlap factor is defined instead by the maximum step movement. The object function $O(x,y)$ and the illumination function $P(x,y)$ are assembled in magnitude and phase providing two images each. Defining such virtual experiment parameters follows what is actually done during a real experiment. Listing 1 in this document shows how a synthetic dataset can be simply produced within the SciComPty framework.
By creating a multi-page complex illumination (M x WIDTH x HEIGHT), one can specify a partially coherent illumination. The output of the simulator for a three-mode illumination (Zernike Polynomials of order), is displayed in the Fig. S1. In this example, the three modes modulates in magnitude and phase a spherical wave, simulating a defocused ptychography setup.

Reconstructions

Many synthetic and real-data experiments have been carried out. We used both workstation and HPC solutions. Table 1 summarises the test configuration.

Simulated data

To reconstruct the simulated dataset, the simulation parameters should be again used during the reconstruction. The simulator is general enough that other softwares can be used to generate a reconstruction. An example reconstruction program for PyNx[9] is the one in Listing 2. The same code can be used also for other reconstructions with the same software. Only the parameters need to be adjusted.
From the simulated dataset, one can obtain the reconstructed object displayed in Fig. S2 and the set of multi-mode illumination shown in Fig. S3. It is important to note that to get such a reconstruction, a number of modes greater than the simulated ones is mandatory. That is why in Fig. S3 there are seven modes. Only the first three are good estimates of the three really employed. A reconstruction with exactly the same amount of modes used during the simulation, fails to provide a good set of $P(x,y)$.

```
# load SciCompty tools
from SciComPtyLibs import simulate, sphericalWave, prepare_obj

# virtual experiment params
OVERLAP = 0.85 # overlap factor
DIMSAMPLE = 1024 # detector size
wavelength = 4.892e-10; # wavelength [m]
fd = 0.8932; # focus-detector distance [m]
fs = 2.5e-3; # focus-sample distance [m]
ps = 13.5e-6; # pixel size [m]

# define pixel size at the sample plane and setup raster grid
delta1 = wavelength * (fd-fs)/(DIMSAMPLE*ps)
maxtrasl = int(DIMSAMPLE - np.floor(OVERLAP*DIMSAMPLE))

# load obj imgs and compose cplx object
imgOrigMod = imgOrigPhase = tifffile.imread('imgs/diatoms_4000.tif')
cplxObj = prepare_obj(imgOrigMod, imgOrigPhase, resizedim=2500)

# create cplx probe P x DIMSAMPLE x DIMSAMPLE
cplxProbe = np.expand_dims(sphericalWave(DIMSAMPLE, wavelength, delta1, -fs,
DIMSAMPLE*delta1/3),0)

# simulate
intensstack, pos = simulate(cplxObj, cplxProbe, fd, fs, ps, DIMSAMPLE, wavelength, maxtrasl)

# save virtual experiment output
tifffile.imsave(outpath + 'diatom_synth_new.tif', intensstack)
np.save(outpath + 'synthpos_new.npy', pos*delta1)
```

Listing 1. SciComPty simulator module API usage example.

CPU	Intel(R) Xeon(R) CPU E5-2643 v4 @ 3.40GHz 24 hyper-threading core, 20 available (virtualisation)
GPU	2x Nvidia Tesla k80, 4 available processors
Virtualisation system	proxmox-ve: 6.1-2 (kernel: 5.3.13-1-pve)
Virtual machine OS	Ubuntu 18.04 LTS (kernel 5.0.0-29-generic)
Python	3.9.5 Anaconda
CUDA	11.3
PyTorch[5]	1.9
SciPy[6]	1.7.1
Scikit-Image[7]	0.18.1
PyNx[8]	2020.2.2

Table 1. System configuration for the algorithm testing.

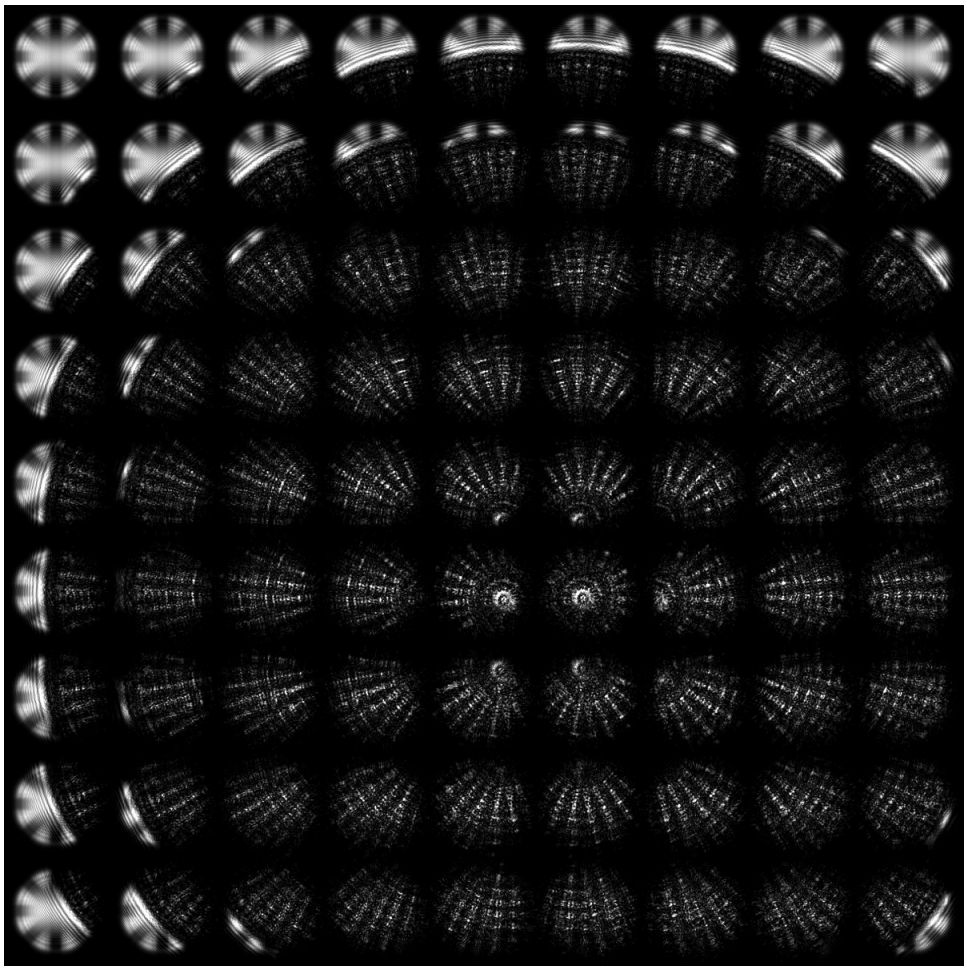

Figure S1. Diffraction pattern (cropped) generated with an image of a diatom and a series of zernike polynomials, modulating a spherical wave.

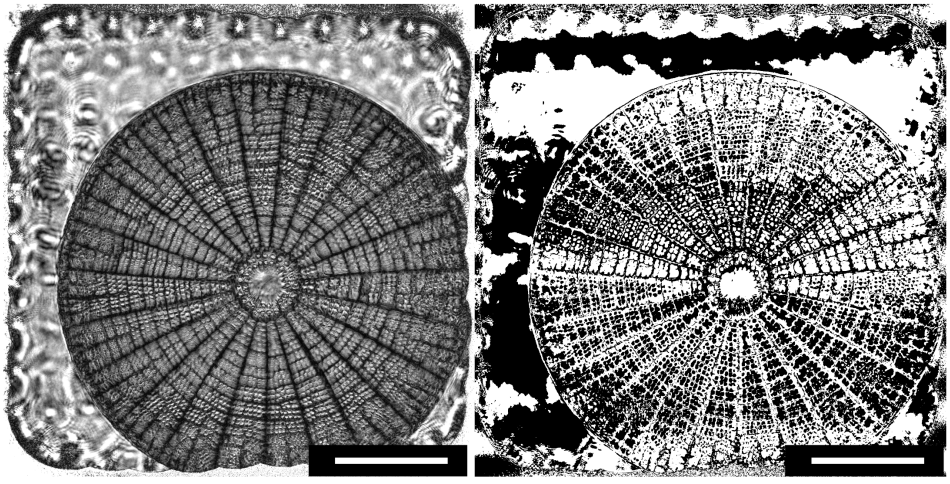

Figure S2. The reconstructed object from the dataset in Fig. S1. The white bar represents 15μm.

```
# import required libraries
import numpy as np
import tifffile as tf
# PyNx tools
from pynx.ptycho import Ptycho, PtychoData, shape, ScaleObjProbe, ML, DM, AP
# SciCompty tools
from SciComPtyLibs import sphericalWave

# load data and positions
diffdata = tf.imread('synthdata/diatom_synth_new.tif')*1.
diffdata[diffdata<0] = 0
positions = np.load('synthdata/synthpos_new.npy')

# reconstruction params
wvl = 4.892e-10; #wavelength
fd = 0.8932; #focal to detector
fs = 2.5e-3; #focal to sample
psize = 13.5e-6; #pixel size

# prepare the reconstruction
distance = fd -fs
pixel_size_object = wvl * (distance)/(diffdata.shape[-1]*psize)
print('Obj plane res: {:.2f} nm'.format(pixel_size_object*1e9))

x,y = positions[:,0], positions[:,1]
x -= x.min()
y -= y.min()
data = PtychoData(iobs=diffdata/diffdata.max(), positions=(x,y),
                  detector_distance=distance, mask=None, pixel_size_detector=psize,
                  wavelength=wvl, near_field=False)

# probe init
nprobes = 7
probeinit = sphericalWave(diffdata.shape[-1], wvl, pixel_size_object, -fs, 6e-6)
probeinit = np.expand_dims(probeinit, 0)
if nprobes >1:
    probeinit = probeinit + 1e-3 * np.random.randn(nprobes, probeinit.shape[-2], probeinit.shape[-1])
    probeinit = probeinit * np.linspace(0.1,1, nprobes)[::-1].reshape(nprobes, 1, 1)

# calculate final obj size
nx0, ny0 = shape.calc_obj_shape(x/pixel_size_object, y/pixel_size_object,
                                (diffdata.shape[-2], diffdata.shape[-1]))
# obj init
obj0 = 0.1* np.exp(1j * np.random.uniform(0, 0.5, (nx0, ny0)))

# main ptycho obj istance
p = Ptycho(probe=probeinit, obj=obj0, data=data, background=None)

# initial scaling
p = ScaleObjProbe() * p

# Optimize
p = AP(update_object=True, update_probe=True, update_pos=False, calc_llk=10, show_obj_probe=10) ** 200 * p
#p = DM(update_object=True, update_probe=True, update_pos=True, calc_llk=100, show_obj_probe=10) ** 400 * p
#p = ML(update_object=True, update_probe=True, update_pos=False, calc_llk=100, show_obj_probe=25) ** 400 * p

# save output
tf.imsave('outobj.tif', p.get_obj())
tf.imsave('outill.tif', p.get_probe())
```

Listing 2. PyNx example code for reconstructions

Note that in the simulated reconstruction of Fig. S2 the raster grid artefact is quite visible. The jitter that has been added to make the raster scan grid less regular is in fact really small. The effect is quite visibile especially for texture-free areas.

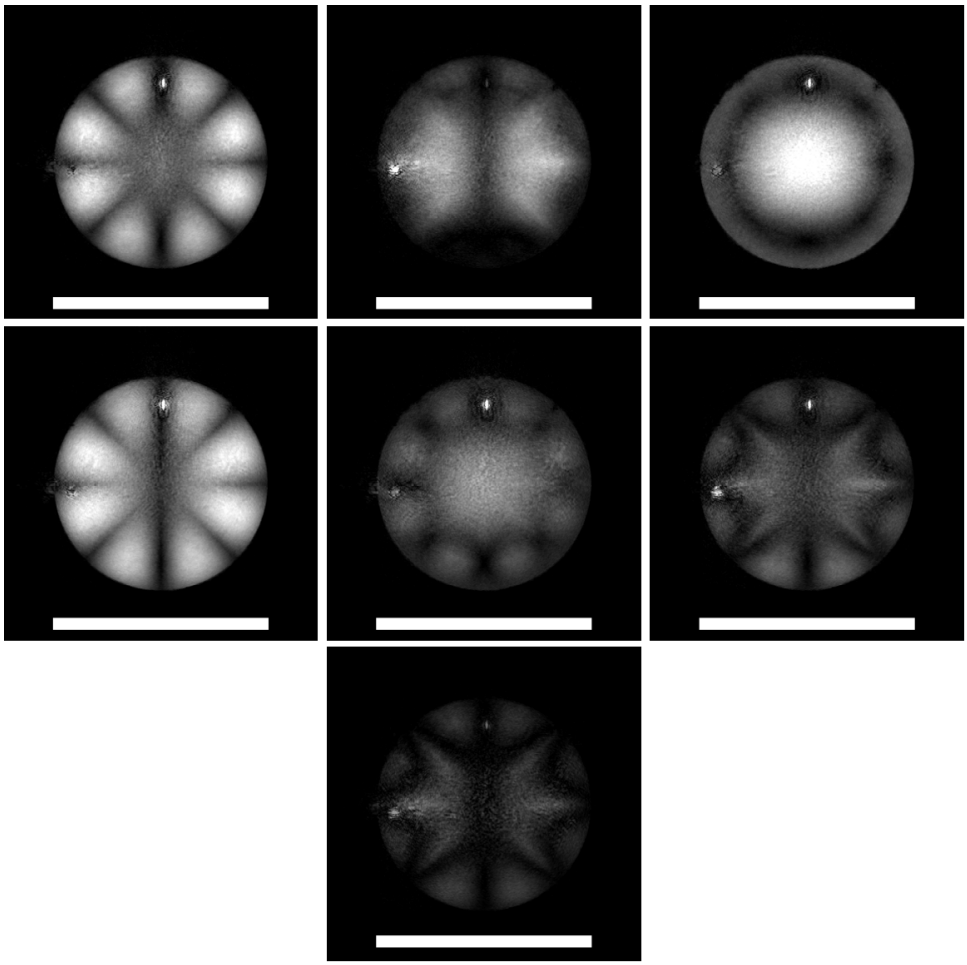

Figure S3. Reconstructions of the 7 modes for the 3-modes diffraction data in Fig. S1. The white bar represents 15μm.

Real soft-X-ray data

In Fig. S4 panel (A) the phase map of a group of chemically fixed mesothelial cells is shown: Mesenchymal-Epithelial Transition (MET) cells were grown in silicon nitride windows and were exposed to asbestos fibres[10]. The absorbing diagonal bar is indeed an asbestos fibre included in the sample. Panels (B) and (C) of figure S4 show the reconstructed illumination in magnitude and phase. The reconstruction is carried out in *SciComPty* for roughly 1000 iteration with position correction. As it can be seen, the entire object box is reconstructed, thanks to the M-rPIE algorithm. In this case a rebin of factor 4 is employed. The code used to obtain such a reconstruction is in List. 3. The same scheme is used for any SciComPty based reconstruction. The reconstructions in figure 3 of the main article (MET cells) are produced with three probe modes. This number of modes has been chosen as a trade-off between reconstruction quality and computational time, that as has been said before, scales linearly with the number of modes. Actually increasing the number of modes did not increase the reconstruction quality. The reconstruction of the P_m modes is displayed in Fig. S5, which shows a well decaying intensity as predicted.
The observable ptychographic resolution gain[11] G_p can be calculated for this reconstruction by the following expression:

$$G_p = \frac{d}{\delta_s} = \frac{9\mu m}{50nm} = 180 \tag{2}$$

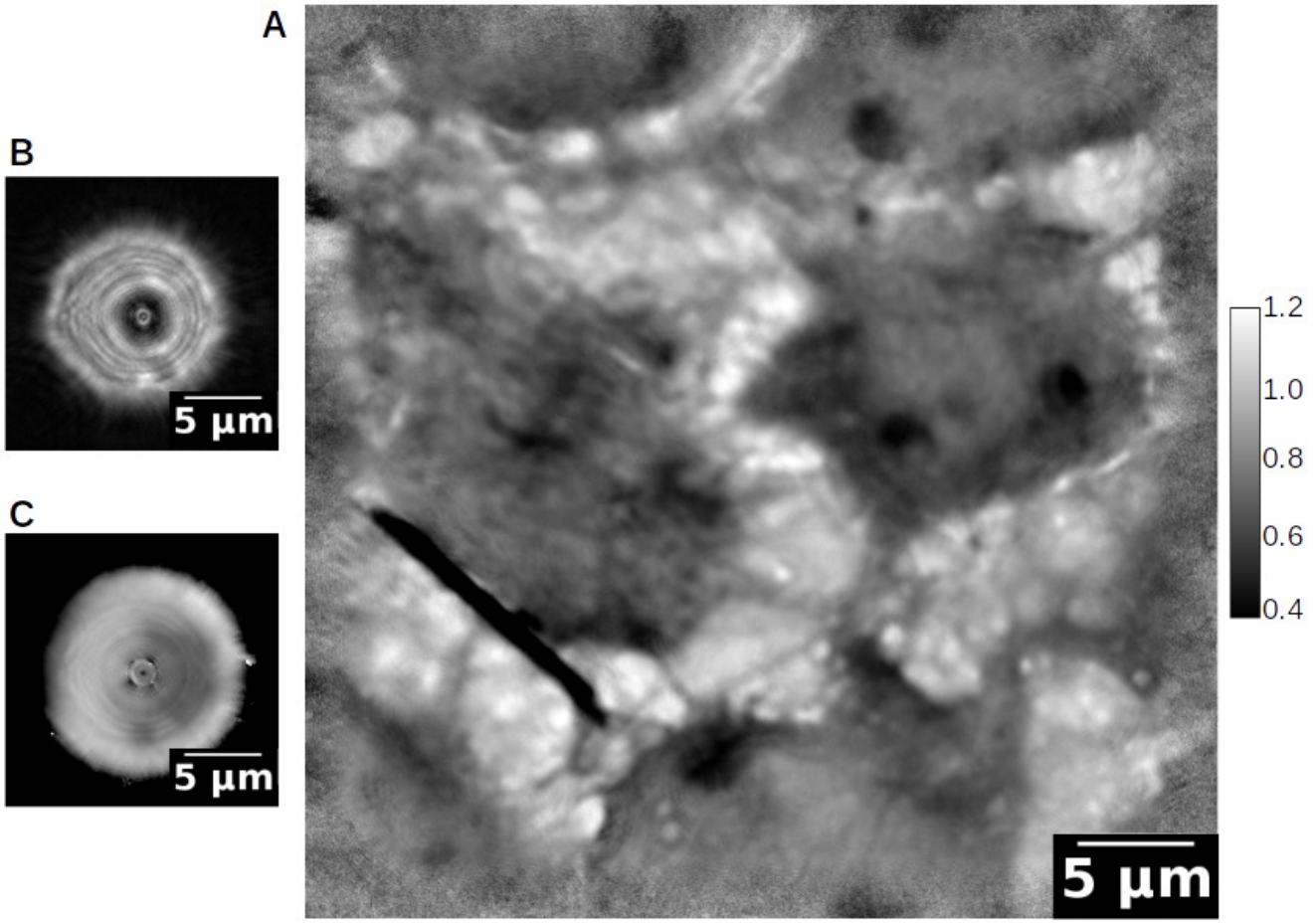

Figure S4. SciComPty phase reconstruction of a MET cells sample grown in silicon nitride windows and exposed to asbestos fibres (the thick bar on the bottom left).

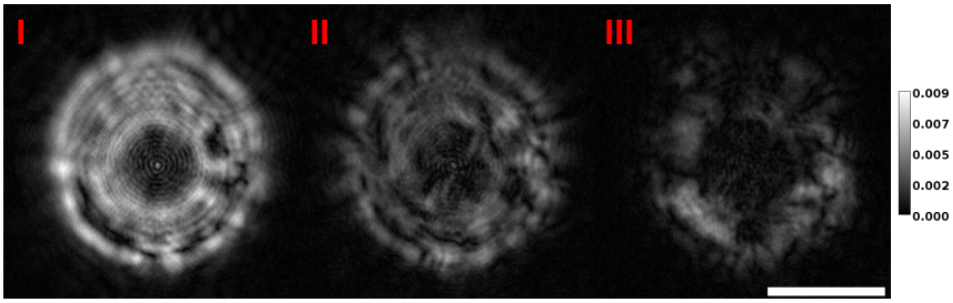

Figure S5. The illumination P(x,y) reconstructed at the sample plane (magnitude) here shows a modal decomposition of 3 modes.

```
# load required libs
import numpy, tifffile, torch, skimage
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")

# SciComPty libs
from SciComPty_libs.ptyobj_tools import PtyRecon, propagator, blur_mask
from SciComPty_libs.generic_tools import composeAgainImage, load_positions, genCircleMask, plot_curr

# params
DIMSAMPLE = 1024
psize = 2*20e-6
fs = 360e-6
en = 1020
distance = 0.7515
wvl = (6.62606957e-34 * 299792458.) / (en * 1.602176565e-19) # wavelength in meter
pixel_size_object = wvl * (distance-fs)/(DIMSAMPLE*psize)

# load data and positions
data = tifffile.imread(datapath)**0.5
data = data/numpy.amax(data)
white = numpy.mean(data, axis=0)
mask = white > 0.5*numpy.amax(white)
pixpos = load_positions(pospath)/pixel_size_object

# probe init
nprobes = 3
probeinit = sphericalWave(DIMSAMPLE, wvl, pixel_size_object, -fs, 6e-6)
probeinit = np.expand_dims(probeinit, 0)
if nprobes >1:
    probeinit = probeinit + 1e-3 * np.random.randn(nprobes, probeinit.shape[-2], probeinit.shape[-1])
    probeinit = probeinit * np.linspace(0.1,1, nprobes)[::-1].reshape(nprobes, 1, 1)

# init full obj
canvas, c,s, xpos, ypos, data = composeAgainImage(data, pixpos, flipIt=False, weightFun=white)
objguess = 0.5*np.ones_like(canvas).astype(np.complex64)

# mask for pos correction
rscanvas = s**0.125
rscanvas -= rscanvas.min()
rscanvas = rscanvas/rscanvas.max()
rscanvas = skimage.transform.resize(rscanvas,
                                    (len(numpy.unique(pixpos[:,0])), len(numpy.unique(pixpos[:,0]))))
rscanvas /= rscanvas.max()

# init illumination
mask = genCircleMask(data.shape[1], data.shape[1] *0.5, 1, 1)

# init pty obj
Ptyobj = PtyRecon(np.fft.fftshift(data), xpos, ypos, fd, fs, ps, wavelength,
                  probeguess=probeguess.astype(np.complex64), objguess=objguess,
                  canvasmask=rscanvas, posupalpha=10., detectormask=None)

# optimize for 200 epochs
for ep in range(200):
    # M-rPIE iteration
    rloss, posloss = Ptyobj.updateRpieMulti(upobj=.5, upill=.5, uppos=True, beta=.5)
    #rloss, posloss = Ptyobj.updateEpieMulti(upobj=.5, upill=.5, uppos=0, beta=.5)
    if ep % 5 == 0:
        obj, ccprobe = Ptyobj.objguess.cpu().numpy()[0], Ptyobj.probeguess.cpu().numpy()
        plot_curr(obj, ccprobe.sum(axis=0), ep, rloss,frac=1/8)

# save output
tifffile.imsave('out_obj.tif', obj); tifffile.imsave('out_ill.tif', ccprobe);
```

Listing 3. SciComPty example code for reconstructions

Post processing

As said in the main text, for many reconstructions obtained with other algorithms (e.g. AP, DM, ML in[9]) not embedded in SciComPty, a post-processing methodology is required. An example is in Fig. S6 panel (a). The real RAW output is shown in Fig. S6 panel (b), where one can only perceive the correct orientation of the picture, by looking at the strong features of the fibre in the right. The visible effect is a strong phase modulation affecting the object. Being in the object space, one way to correct this artefact is to shift its complex Fourier transform in the Fourier space, by employing the shifting property of the transform. To do so, we can again use the phase correlation algorithm to find the cross-correlation peak among the two transforms. Fig. S6 panel (c) shows the magnitude of the Fourier Transform for the two reconstructions in panel (a) and (b), here stacked and colored in red and green. An offset of roughly 20 pixel is visible. Figure 3 of the main text shows the corresponding reconstruction obtained with SciComPty, which appears way less blurry.

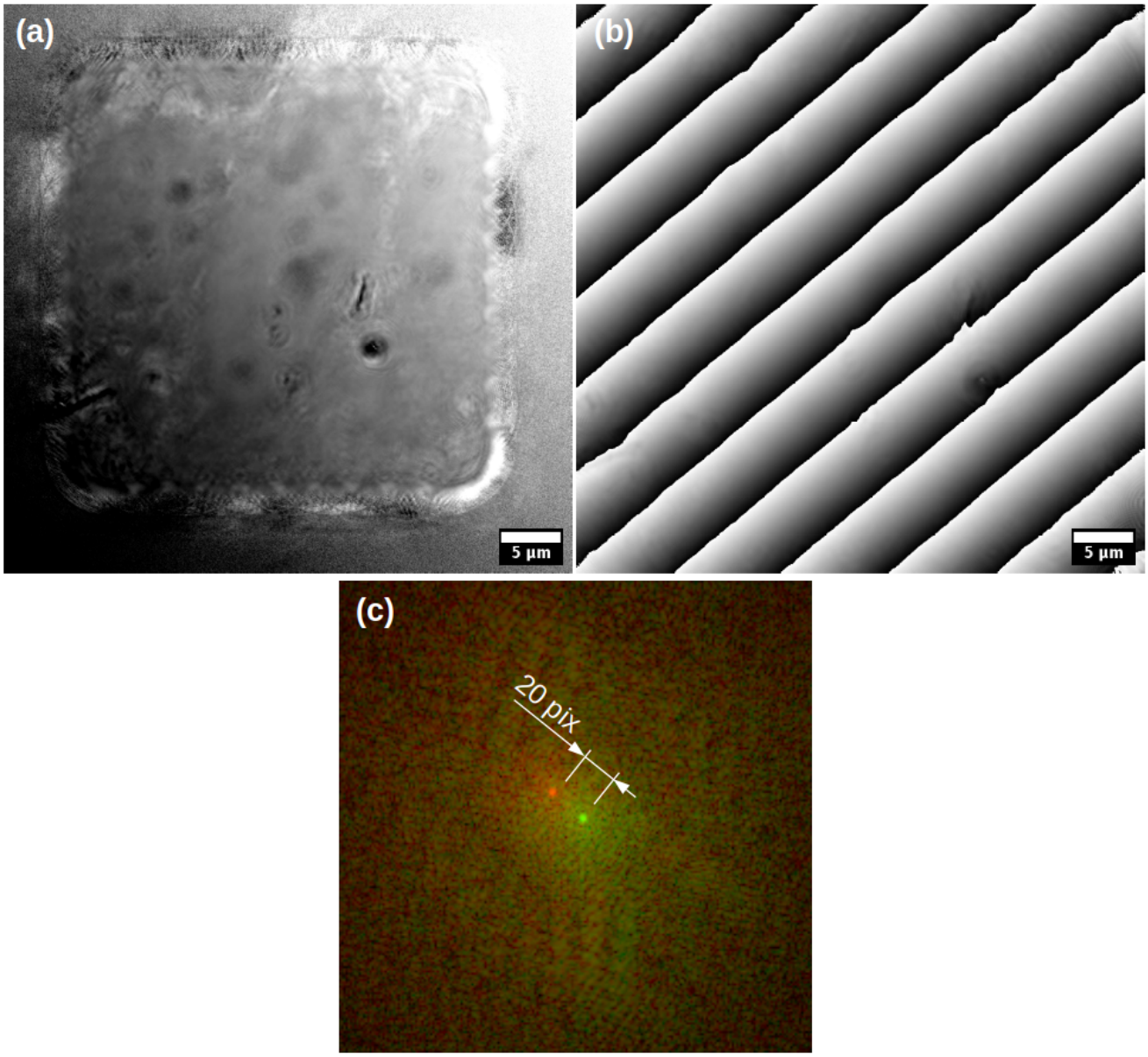

Figure S6. Post-processing required for the reconstructions obtained with[9]. Panel (a) is the final phase; panel (b) is the raw phase image; panel (c) shows a two-channel image where in red we display the raw object Fourier Transform, while in green the post-processed. The phase of the ifft of panel (c) provides panel (a) for the green channel and panel (b) for the red channel.

To verify the behaviour for other datasets, we employed an optical ptychography dataset which has been released in[12]. Again, a similar artefact can be found, see Fig. S7 panel(a). The same set of diffractions and parameters is given also to the M-rPIE recipe in SciComPty, producing the phase map in panel (b). Differently from the previous case, here the main culprit is a shifted illumination, as can be seen confronting panel (c) with panel (d) of Fig S7.

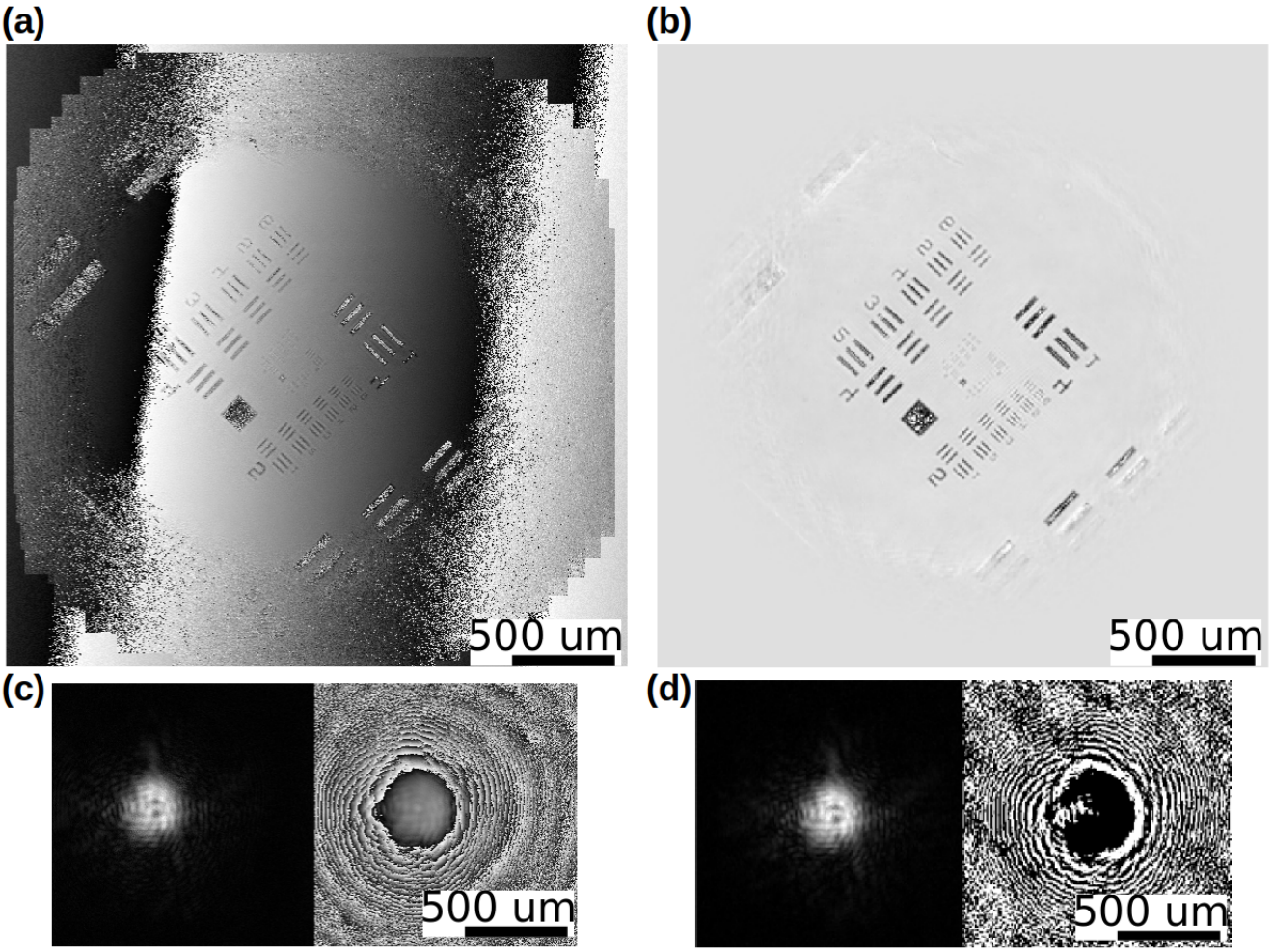

Figure S7. The illumination P(x,y) reconstructed at the sample plane (magnitude) here shows a modal decomposition of 3 modes.

Dynamics of the position refinement signal

Where to "sense" for the position error? Referring to Fig. S8, for each *jth* $(x,y)_j$ position, many methods[13–15] observe the 2D phase/cross-correlation or at A) the object plane (between a corrected and uncorrected object estimate at the same *jth* computational box, $XCORR_A$), or at B) on the detector plane, calculating the correlation between the *jth* acquired and simulated diffraction pattern, $XCORR_B$. This latter methodology is used for example in CT[16]. Also, it seems unnatural to check between two "artificial" quantities ($\mathbf{o_j}$ and $\mathbf{o'_j}$) instead of relying at least on one real image (the acquired diffraction pattern).
To investigate the dynamics of the error signal, we used the SciComPty simulator to generate a ptychography dataset in which the object function $O(x,y)$ and the illumination function $P(x,y)$ are assembled in magnitude and phase, providing two images each. The $O(x,y)$ function is made up by the "Perseverance" Mars Rover and the famous "Bacteria" test image, while the smaller $P(x,y)$ is composed by "Chelsea" (the cat) and "Astronaut". Then the positions are perturbed by adding a Gaussian error on both the x,y coordinate, with a standard deviation equal to 5% of each object view. A reconstruction is started enabling the position correction routine, based on the error signal $XCORR_A$ or $XCORR_B$. The process is monitored iteration after iteration by recording: 1) the reconstruction error (blue) as defined in[17] (Eq. 3; 2) the mean average error as estimated by the

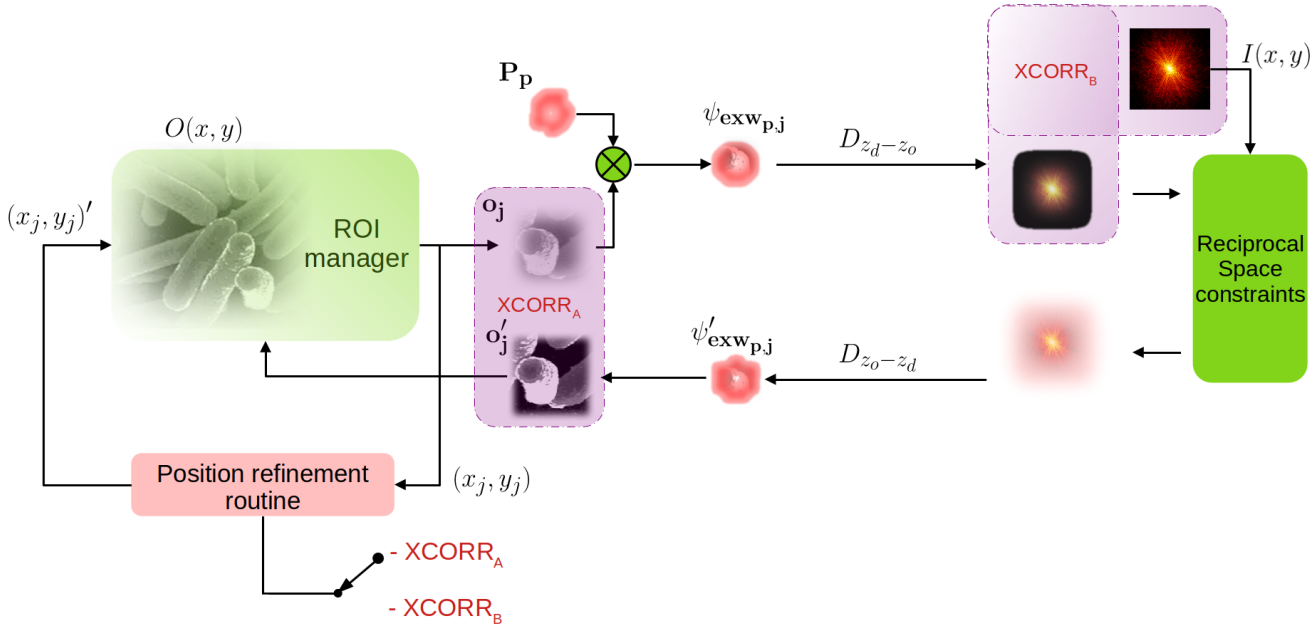

Figure S8. Position refinement routine integrated into a PIE algorithm; the position error signal (argmax of 2D cross-correlation) can be calculated meaningfully in two points, $XCORR_A$ or $XCORR_B$ (purple boxes). The estimated positions (x,j) are then updated to $(x_j,y_j)'$ iteration by iteration.

$XCORR_A$ or $XCORR_B$ procedure (green, Eq. 4) and 3) the ground truth average error (orange) between the real and the current position vector (Eq. 5). Also the diffraction patterns are corrupted by a small amount of gaussian noise. The reconstruction is carried out for 300 iterations. The reconstruction error is calculated as the L2-norm of the difference between $o_j(x,y) = \mathbf{o_j}$ and $o'(x,y) = \mathbf{o'_j}$:

$$E = ||o(x,y) - o'(x,y)||^2 \tag{3}$$

The position estimation is based on the normalised cross correlation between two 2D arrays $a(x,y) = \mathbf{a}$ and $b(x,y) = \mathbf{b}$ having width of $2W+1$ pixels and height of $2H+1$ pixels. We calculate the coordinates of the cross correlation peak between the 2D arrays (choosed at position $XCORR_A$ or $XCORR_B$, to check for a translation shift. From these coordinates one can calculate the length of this vector:

$$\varepsilon_{pos} = \sqrt{||\underset{x,y}{\operatorname{argmax}}\{a(x,y) * b(x,y)\}||^2} = \sqrt{||\underset{x,y}{\operatorname{argmax}}\{\sum_{k=-H}^{H}\sum_{l=-W}^{W} a(x,y)\cdot b(x+k,y+l)\}||^2} \tag{4}$$

remembering that these arrays are different if using $XCORR_A$ or $XCORR_B$, as previously described (see Fig. S8). This signal is used to correct the position.
The ground truth position error is calculated as the mean of the distance between any position; given $\mathbf{f}_j = [x_j,y_j]^T$ as the *jth* 2D ground truth position vector and $\mathbf{g_j}$ as its estimation refined by the algorithm, we define the distance between them as:

$$d_j = \sqrt{||\mathbf{f_j} - \mathbf{g_j}||^2}. \tag{5}$$

The corresponding mean provides the ground truth position error ε_g:

$$\varepsilon_g = \frac{1}{J}\sum_{j=1}^{J} d_j \tag{6}$$

where J is the number of positions vectors and diffraction patterns in the dataset. As said in the main text, the estmated sub-pixel position error is amplified and used to guide the correction; in the proposed method we use a dynamic gain factor η, independently for the x and y coordinates, producing new coordinates x'_j, y'_j

$$x'_j = x_j + \eta_{x,j} \cdot \underset{x}{\text{argmax}}\{XCORR_{i\in[A,B]}\}, \tag{7}$$

$$y'_j = y_j + \eta_{y,j} \cdot \underset{y}{\text{argmax}}\{XCORR_{i\in[A,B]}\}, \tag{8}$$

If η is a constant equal to 1 and $XCORR_A$ is chosen, the output of the reconstruction can be seen in Fig. S9: the estimated position error (Eq. 4) signal (green curve) drops rapidly to zero, meaning that no further correction on the positions will be done. As reported in[15], the same effect can be observed for a constant gain. That is why both the reconstruction error (blue) and the ground truth position error (orange, Eq. 5) converge rapidly but to a high value, denoting a bad reconstruction. Also for the case of $XCORR_B$ (Fig. S10, only a marginally better reconstruction (see the inset panel of the illumination magnitude) can be gathered. The error signal (green curve in S9 and S10) is noisier, due to the fact that only half of the information (real images) is present.

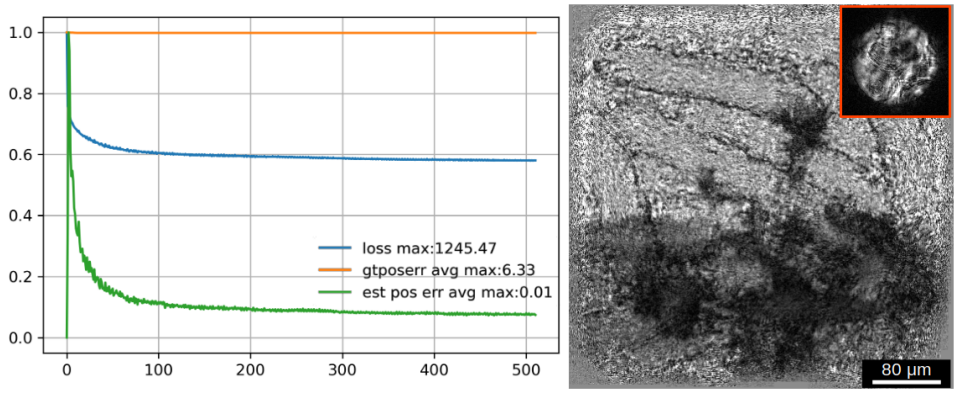

Figure S9. XCORRA signal (green curve) with $\eta = 1$ has a low dynamic range, practically not refining the coefficients. The eye of the cat (illumination magnitude) is not correctly retrieved.

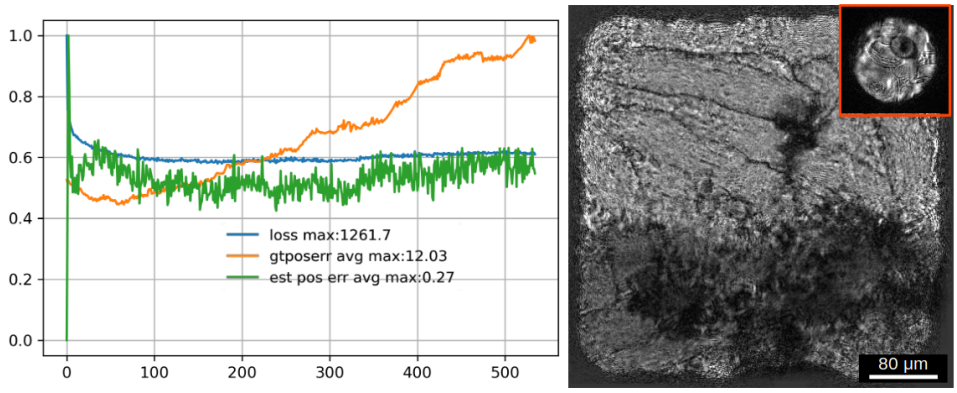

Figure S10. XCORRB signal (green) is so noisy that can make the error diverge also for $\eta = 1$. The reconstruction is marginally better than the one in Fig. S9.

By introducing a gain factor ($\eta > 1$) the convergence can be greatly improved not only in speed terms but also toward a better solution (hopping local minima). An adaptive correction is required, ideally per-parameter.

Figures S11 shows how the implementation of Adam is effective in adapting the gain of the error signal, retrieved at XCORRA, just where in the case of unity gain the signal was ineffective. Conversely, applying Adam at XCORRB (Fig. S12) the correction is marginally improved due to its noisier nature.

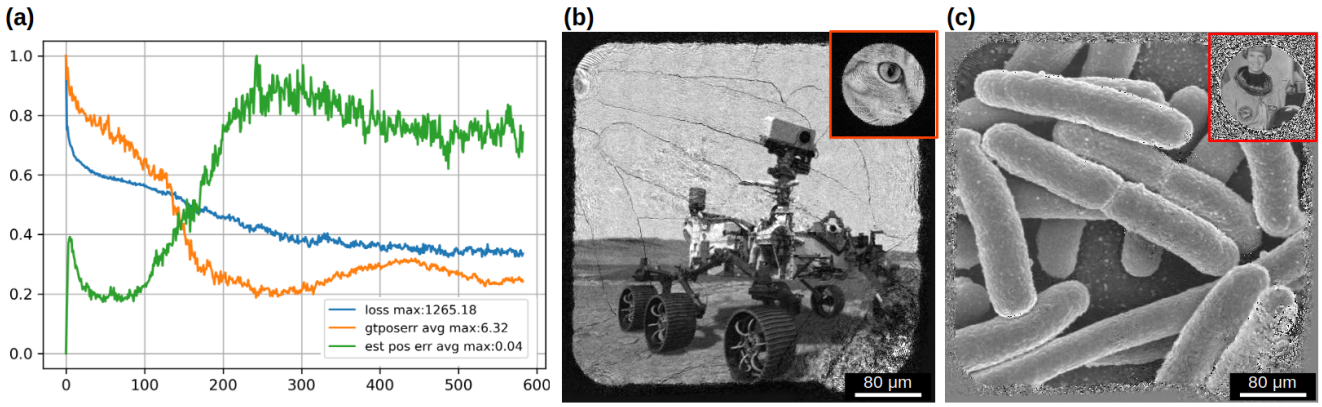

Figure S11. Adam is now used as a gain controller, in conjunction with XCORRA (green). The loss (blue) and the ground truth position error (orange) are reduced, providing a good reconstruction for the magnitude and phase of both the object (panel B and C) and the illumination (insets in B and C).

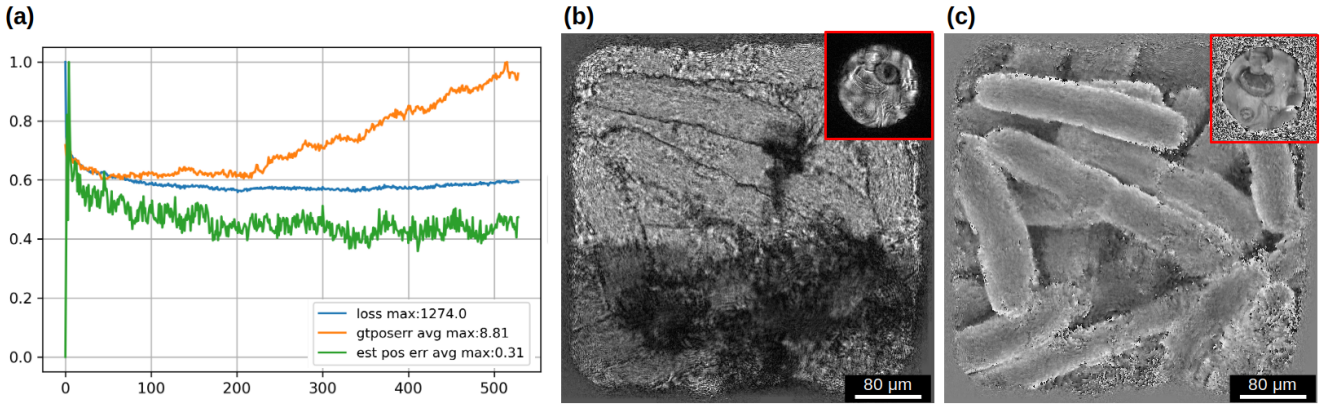

Figure S12. Differently from Fig. S11, the error signal in XCORRB is noisy allowing only a partial correction.

The choice of an error signal from the object plane finally coincides with what was proposed initially in the literature[13–15], but only if a gain factor is applied; the statistic-based method proposed in[15], in our experiments and implementation, not only is somehow difficult to tune but does not provide a vector of per-parameter controlled acceleration.

Position refinement methods on simulated data

In this section, we test the three position refinement methods with simulated data. The algorithm of choice here is ePIE. Two different degree of noise are corrupting the positions. In Fig. S13 a small amount of noise (range of 7 pixels) is added to the position vectors; panel (a), (b) and (c) refers to the proposed method; panel (d), (e) and (f) show the output of the method in[15], again implemented in SciComPty, while panel (g), (h) and (i) display the output of the position correction method in[9]. Similarly to the other graphs, the curves plotted in the first column are respectively the reconstruction error (blue), the ground-truth position error and (orange) and the estimated error by the algorithm (green). Panel (g) is different as the quantities we provide as metrics are not available in the PyNx software. That is why we show the two available quantities, Log Likelihood "LLK" as a measure of the error of the reconstruction, and the estimated position error. As can be seen, the proposed method provide the best reconstruction, with a correction factor that is quite uniform in the entire reconstruction. For the same corruption, the method in[15] (2nd row) has an accuracy of roughly half of the proposed method; confronting the curves in panel (a) and

(d), in the very first iterations the correction is extremely large, then it stops. We tried to tweak a bit the parameters of the method, but in the end we ended up using the values proposed in[15] as it were already providing the best results. Unfortunately the method in[9] fails in providing a usable correction; indeed, the correction strength dissipates early and no further action is practically taken after iteration 25. In this particular scenario, changing the reconstruction method to DM[3] or ML[18] makes the reconstruction even worse, as the optimiser struggle to find the correct illumination. The same problem was experienced also in real-data reconstruction (see main text). Now a larger random error is added to the vectors. The output of the experiments are show in Fig. S14. Panel (a,b,c) show the output of the proposed method, while (d,e,f) refer to the algorithm in[15]. With a range of 14 pixel here we are at more the 10% of a 128 pixel wide diffraction pattern. Only the proposed algorithm is able to partially correct the positions, providing an usable reconstruction. The method in[15] give up after few iterations.

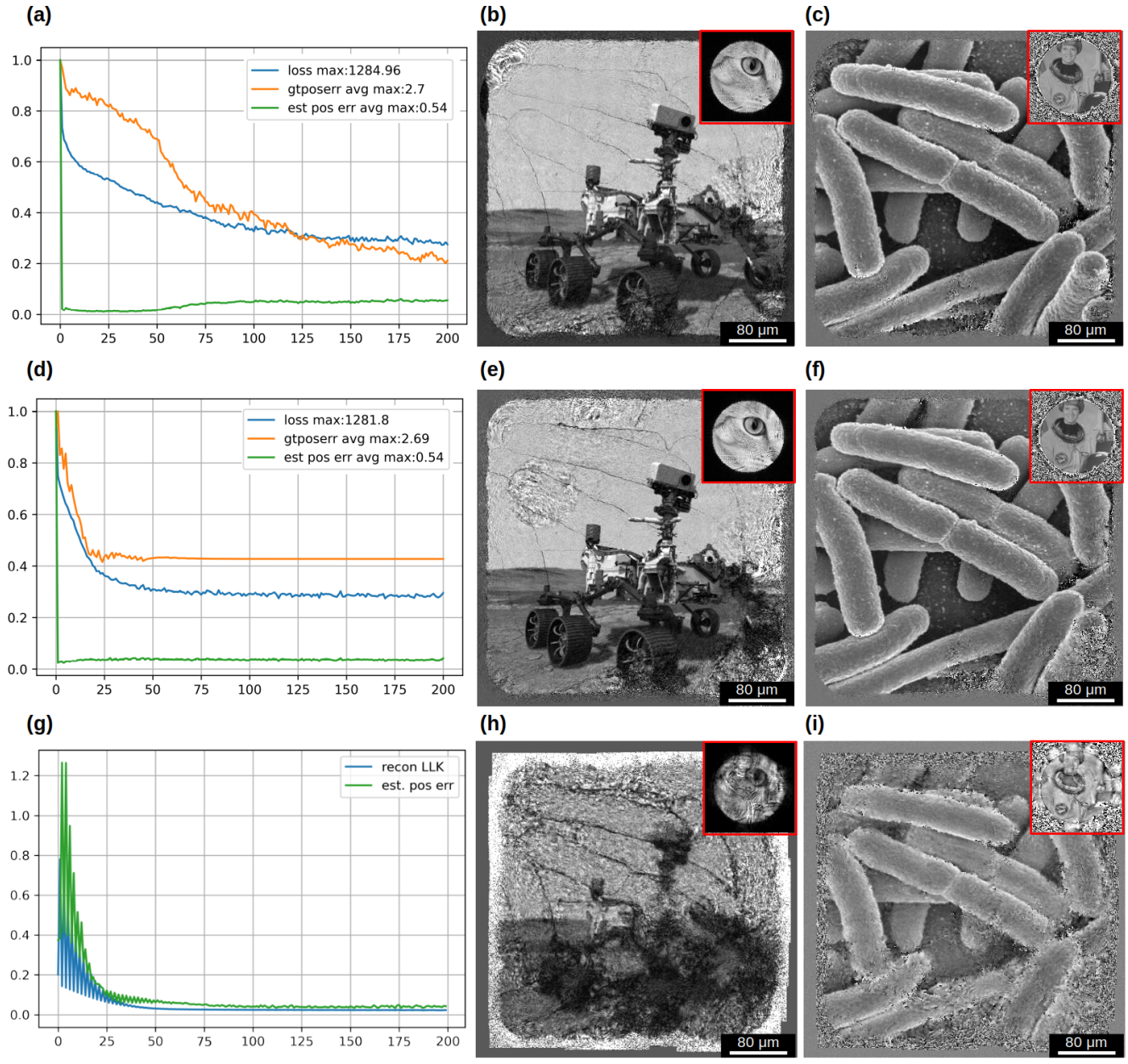

Figure S13. Comparison of the position refinement methods; panel (a,b,c): Proposed Adam method; panel(d,e,f) method in[15]; panel (g,h,i): method in[9]

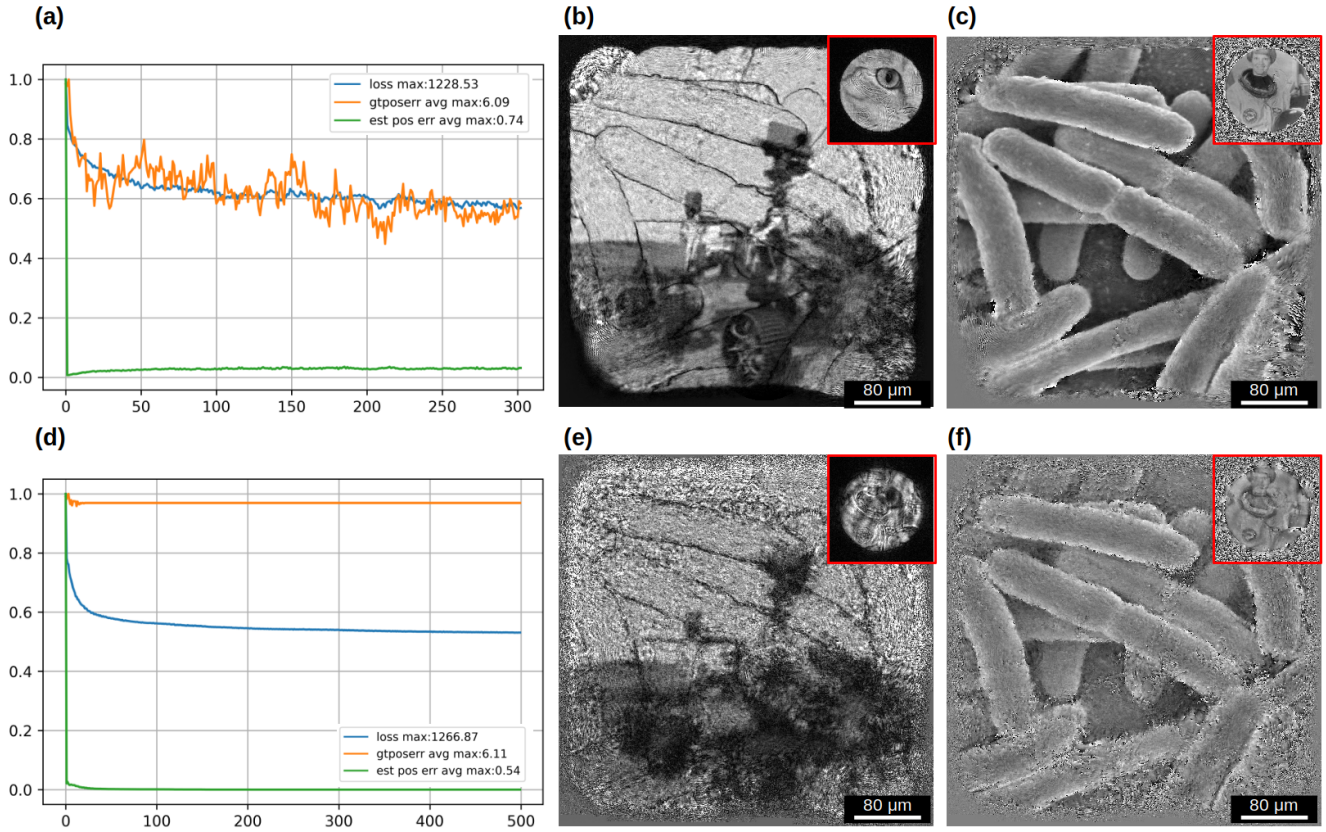

Figure S14. Comparison of the position refinement methods; panel (a,b,c): Proposed Adam method; panel(d,e,f) method in[15]; panel (g,h,i): method in[9]

References

1. Schmidt, J. D. *Numerical simulation of optical wave propagation with examples in MATLAB* (SPIE, Bellingham, Washington, 2010).
2. Dierolf, M. *et al.* Ptychographic coherent diffractive imaging of weakly scattering specimens. *New J. Phys.* **12**, 035017, DOI: 10.1088/1367-2630/12/3/035017 (2010).
3. Thibault, P. *et al.* High-resolution scanning x-ray diffraction microscopy. *Science* **321**, 379–382, DOI: 10.1126/science.1158573 (2008). https://science.sciencemag.org/content/321/5887/379.full.pdf.
4. Edo, T. B. *et al.* Sampling in x-ray ptychography. *Phys. Rev. A* **87**, 053850, DOI: 10.1103/PhysRevA.87.053850 (2013).
5. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M. *et al.* (eds.) *Advances in Neural Information Processing Systems 32: NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 8024–8035 (2019).
6. Jones, E., Oliphant, T., Peterson, P. *et al.* SciPy: Open source scientific tools for Python (2001–).
7. van der Walt, S. *et al.* scikit-image: image processing in Python. *PeerJ* **2**, e453 (2014).
8. Favre-Nicolin, V. *et al.* *PyNX*: high-performance computing toolkit for coherent X-ray imaging based on operators. *J. Appl. Crystallogr.* **53**, 1404–1413, DOI: 10.1107/S1600576720010985 (2020).
9. Mandula, O., Elzo Aizarna, M., Eymery, J., Burghammer, M. & Favre-Nicolin, V. *PyNX.Ptycho*: a computing library for X-ray coherent diffraction imaging of nanostructures. *J. Appl. Crystallogr.* **49**, 1842–1848, DOI: 10.1107/S1600576716012279 (2016).
10. Cammisuli, F. *et al.* Iron-related toxicity of single-walled carbon nanotubes and crocidolite fibres in human mesothelial cells investigated by synchrotron xrf microscopy. *Sci. Reports* **8**, 706, DOI: 10.1038/s41598-017-19076-1 (2018).

11. Jacobsen, C., Deng, J. & Nashed, Y. Strategies for high-throughput focused-beam ptychography. *J. Synchrotron Radiat.* **24**, 1078–1081, DOI: 10.1107/S1600577517009869 (2017).

12. Enders, B. & Thibault, P. A computational framework for ptychographic reconstructions. *Proc. Royal Soc. A: Math. Phys. Eng. Sci.* **472**, 20160640, DOI: 10.1098/rspa.2016.0640 (2016). https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.2016.0640.

13. Tripathi, A., McNulty, I. & Shpyrko, O. G. Ptychographic overlap constraint errors and the limits of their numerical recovery using conjugate gradient descent methods. *Opt. Express* **22**, 1452–1466, DOI: 10.1364/OE.22.001452 (2014).

14. Dwivedi, P., Konijnenberg, A., Pereira, S. & Urbach, H. Lateral position correction in ptychography using the gradient of intensity patterns. *Ultramicroscopy* **192**, 29–36, DOI: https://doi.org/10.1016/j.ultramic.2018.04.004 (2018).

15. Zhang, F. *et al.* Translation position determination in ptychographic coherent diffraction imaging. *Opt. Express* **21**, 13592–13606, DOI: 10.1364/OE.21.013592 (2013).

16. Gürsoy, D. *et al.* Rapid alignment of nanotomography data using joint iterative reconstruction and reprojection. *Sci. Reports* **7**, 11818, DOI: 10.1038/s41598-017-12141-9 (2017).

17. Reinhardt, J. & Schroer, C. Quantitative ptychographic reconstruction by applying a probe constraint. *J. Instrumentation* **13**, C04016–C04016, DOI: 10.1088/1748-0221/13/04/c04016 (2018).

18. Thibault, P. & Guizar-Sicairos, M. Maximum-likelihood refinement for coherent diffractive imaging. *New J. Phys.* **14**, 063004, DOI: 10.1088/1367-2630/14/6/063004 (2012).