Simulator Documentation: `tem26.py`

Petar

October 24, 2021

Contents

1 Introduction

In this document, I will describe the usage of the TEM simulator `tem26.py` and the equations which underlie its calculations. There may be features in the code which are not described in this document, however.

Basically anywhere that a distance is to be specified by the user, the relevant unit will be Å. Angles may be in degrees or radians. It will hopefully be obvious, or at least easily seen in the comments of the script.

2 Physics

2.1 Constants

A number of constants are specified in the dictionary `const`:

```
const = {
    'alpha': 0.0072973525693, # fine structure constant
    'c': 299792458, # speed of light [m/s]
    'hbar': 6.62607004e-34/(2*np.pi), # reduced planck const [kg*m^2/s]
    'e_mass': 0.511e6, # electron mass [eV/c^2]
    'e_compton': 2.42631023867e-2, # compton wavelength [A]
    'n_avogadro': 6.02214076e23, # avogadro's number [mol^-1]
    'a0': 0.529, # bohr radius [A]
    'e_charge_VA': 14.4, # electron charge [V*A]
    'e_charge_SI': 1.602176634e-19, # elementary charge [C]
    'J_per_eV': 1.602176634e-19 # joules per eV (unit conversion)
    }
```

Additionally, the dictionaries `z_dict` and `vdw_dict` contain the atomic numbers and van der Waals radii [https://en.wikipedia.org/wiki/Van_der_Waals_radius] of the elements, respectively. Note that these are used to read files from the Protein Data Bank and create atomic potentials, and that missing elements will need to be added manually into these hardcoded dictionaries. Additionally, the file `fparams.csv` contains the coefficients of a 12-parameter function for each element, which approximates its atomic potential (Sec. 2.5.2). This file does not need to be updated, but it does need to be kept together with `tem#.py`.

2.2 Derivative parameters

A number of useful parameters are not regarded as fundamental. They are automatically determined by the function `get_deriv_params` based on "more fundamental" inputs:

$$\begin{aligned}
\gamma_L &= 1 + \frac{V_{HT}}{m_e} && \text{Lorentz factor} \\
\beta_e &= \sqrt{1 - \gamma_L^{-2}} && \text{electron velocity} \\
\lambda_e &= \lambda_C / \gamma_L \beta_e && \text{electron wavelength} \\
\sigma_e &= \frac{2\pi}{\lambda V_{HT}} \left(\frac{m_0^2 + eV_{HT}}{2m_0^2 + eV_{HT}} \right) && \text{interaction parameter} \\
s_{Ny} &= 1/2p && \text{Nyquist frequency} \\
w_0 &= \lambda_L / N_A && \text{laser waist radius} \\
z_R &= \lambda_L / N_A^2 && \text{laser Rayleigh range}
\end{aligned}$$

Note the unusual subscripts on γ_L and β_e. These exist because γ is the aberration function of the objective lens (Sec. 2.4.1) and β is the electron beam tilt angle (Sec. 2.4.3).

2.3 Image formation

Image formation describes the propagation of an *exit wave* $\psi_{op}(\mathbf{x})$ present in the object plane of the microscope through the TEM column. This is carried out by `TEM`, which converts `exit_wave_op` to `psi_ip` in several steps. This output is then converted to a probability density, and finally to an image, as described below.

A 2D Fourier transform of the exit wave brings the wave to the back focal plane (`propagate_op_to_bfp`), where it is multiplied by the transfer function $H(\mathbf{s})$ (`apply_transfer_func`) to arrive at the back focal plane wavefunction $\psi_{bfp}(\mathbf{s})$:

$$\psi_{bfp}(\mathbf{s}) = \mathcal{F}[\psi_{ip}(\mathbf{x})](\mathbf{s}) H(\mathbf{s}) \tag{1}$$

Note that the transfer function is generally called `H_bfp` in the code, but I'll omit the subscript in this document.

The wavefunction is then inverse Fourier transformed (`propagate_bfp_to_ip`) to obtain the wavefunction in the image plane, $\psi_{ip}(\mathbf{x})$. If you're bothered by using the same coordinate system in the object and image plane, imagine that the microscope has a magnification of 1 and that image plane fields are written in a coordinate system $\mathbf{x}'$, with the prime denoting the object-image correspondence with the object space $\mathbf{x}$ as in a proper optics textbook. The prime will be implicit here.

$$\psi_{ip}(\mathbf{x}) = \mathcal{F}^{-1}[\psi_{bfp}(\mathbf{s})](\mathbf{x}) \tag{2}$$

The camera measures intensity which is corresponds to the probability density. We will calculate the square modulus of the wavefunction (`calc_image_pdf`):

$$I_{ip}(\mathbf{x}) = |\psi_{ip}(\mathbf{x})|^2 \tag{3}$$

In the code, the output of `calc_image_pdf` is `img_ip`, an array which has a value of 1 in each pixel in the absence of an object. In other words, if the pixel value is > 1 then electrons are more likely to arrive there, and if it's < 1 then they are less likely to arrive there.

Given a pixel size p (side length, in Å) and a total dose n_e (in $e^-/\text{Å}^2$), the mean number of electrons in each pixel is given by

$$\mu_e(\mathbf{x}) = n_e p^2 I_{ip}(\mathbf{x}) \tag{4}$$

and a noisy image which has electron counts $c_e(\mathbf{x})$ is generated as a Poisson realization

$$c_e(\mathbf{x}) \sim \text{Pois}[\mu_e(\mathbf{x})] \tag{5}$$

The function which performs these operations is `pdf_to_image`. If the option `noise_flag=False`, the output will correspond to Eq. 4, and if it's `True` then the output will correspond to Eq. 5.

2.4 The transfer function, H

The transfer function is calculated using `get_transfer_func` (with `return_ctf=False`, which is the default setting). The transfer function is broken down according to

$$H(\mathbf{s}) = A(\mathbf{s})e^{i\xi(\mathbf{s})} \tag{6}$$

If the effects of partial coherence are to be ignored (`scope['coherence_envelopes_on']=False`), then

$$A(\mathbf{s}) = 1 \tag{7}$$
$$\xi(\mathbf{s}) = -\chi(\mathbf{s}) \tag{8}$$

where $\chi(\mathbf{s})$ corresponds to the phase in the back focal plane. This function contains contributions from the *objective aberration function* $\gamma(\mathbf{s})$ as well as the *phase plate phase function* $\eta(\mathbf{s})$:

$$\chi(\mathbf{s}) = \gamma(\mathbf{s}) + \eta(\mathbf{s}) \tag{9}$$

The role of partial coherence will be considered in Sec. 2.4.3.

2.4.1 The objective aberration function, γ

The aberration due to the objective is calculated with `get_gamma` and depends upon the defocus f_0 and parameters of the microscope (such as spherical aberration C_s and laser wavelength λ_e). Strictly speaking, the astigmatism may be regarded as a property of the microscope, and could thus arguably belong in `scope`, but astigmatism is adjusted by setting its magnitude f_a with `img['astig_z']`, in the same dictionary where the defocus f_0 is set with `img['mean_z']`. The astigmatism axis ϕ_a is set with `img['astig_angle']`.

With the sign convention that $f_0 > 1$ corresponds to *underfocus*, we have

$$\gamma(\mathbf{s}; f_0) = \pi\left[\frac{1}{2}C_s\lambda_e^3 s^4 - f(\mathbf{s}; f_0)\lambda_e s^2\right] \tag{10}$$

The arguments of $\gamma(\mathbf{s}; f_0)$ are written so as to suggest that the function is evaluated in the back focal plane ($\mathbf{s}$) but is parametrized by f_0, which will become important in the partial coherence calculations of Sec. 2.4.3. I will call the function $f(\mathbf{s}; f_0)$ the *defocus coordinate*, which is calculated with `make_defocus_coord`:

$$f(\mathbf{s}; f_0) = f_0 + f_a\cos\left[2\left(s_\varphi - \phi_a\right)\right] \tag{11}$$

where $s_\varphi = \text{atan2}(s_y, s_x)$ is the azimuthal coordinate in the $\mathbf{s}$ plane, derived from the Cartesian coordinates (s_x, s_y), and generally termed phi and obtained from make_azimuthal_coord. The radial coordinate is given by

$$|\mathbf{s}| = s = s_\rho \tag{12}$$

This coordinate, as well as the Cartesian coordinates (s_x, s_y), are stored in the dictionary S as S['r'], S['x'], and S['y']. This dictionary is obtained from make_bfp_coords_S. The units of these (non-angular) spatial frequency coordinates are Å^{-1}.

The gradient of γ can be calculated analytically with get_gamma_gradient. It is returned in the Cartesian basis:

$$\frac{\partial \gamma(\mathbf{s}; f_0)}{\partial \mathbf{s}} = \frac{\partial \gamma}{\partial s_x}\hat{\mathbf{s}}_x + \frac{\partial \gamma}{\partial s_y}\hat{\mathbf{s}}_y \tag{13}$$

$$= 2\pi \left[(C_s \lambda_e^3 s_\rho^3 - f_0 \lambda_e s_\rho)\cos s_\varphi - f_a \lambda_e s_\rho \cos(s_\varphi - 2\phi_a)\right]\hat{\mathbf{s}}_x + 2\pi \left[(C_s \lambda_e^3 s_\rho^3 - f_0 \lambda_e s_\rho)\sin s_\varphi + f_a \lambda_e s_\rho \sin(s_\varphi - 2\phi_a)\right]\hat{\mathbf{s}}_y \tag{14}$$

2.4.2 The phase plate phase function, η

The phase plate may be any function $\eta(\mathbf{s})$, but we will consider the laser phase plate and ("true") Zernike phase plate.

2.4.2.1 Laser phase plate

The phase shift due to the standing wave created by the laser phase plate (LPP) can be calculated with get_laser_phase. It can be written

$$\eta(L_x, L_y) = \frac{\eta_0}{2}\exp\left(-\frac{2L_y^2}{1+L_x^2}\right)\frac{1}{\sqrt{1+L_x^2}} \times \left\{1 + (1 - 2\beta_e^2\cos^2\theta_p)\exp\left[-\frac{2\theta_t^2}{N_A^2}(1+L_x^2)\right] \times \left(\frac{1}{1+L_x^2}\right)^{1/4}\cos\left[\frac{2L_x L_y^2}{1+L_x^2} + \frac{4L_x}{N_A^2} - \frac{3}{2}\arctan(L_x) - \phi_L\right]\right\} \tag{15}$$

where N_A is the numerical aperture of the cavity, β_e is the velocity of the electrons in units of c, θ_p is the polarization angle of the laser relative to the electron beam axis, θ_t is the angle of out-of-plane tilt of the *laser* beam (i.e. in the plane defined by the direction of laser propagation, $\hat{\mathbf{L}}_x$, and the axis of the microscope, $\hat{\mathbf{z}}$), and ϕ_L is a phase term which equals 0 if the longitudinal cavity mode has an antinode at $L_x = 0$. The term η_0 corresponds to the peak phase shift. More on this in a moment.

The laser polarization angle θ_p can be set to the relativistic reversal angle

$$\theta_{rra} = \arccos\left(1/\sqrt{2}\beta_e\right) \tag{16}$$

by setting laser['pol_angle']='rra'. The reversal angle is calculated with get_reversal_angle.

(L_x, L_y) is a dimensionless coordinate system obtained as the dictionary keys L['x'] and L['y'] from make_laser_coords_L. The laser propagates along L_x, which can be converted to physical units by multiplication with the Rayleigh range z_R. The transverse axis is L_y, converted to physical units by multiplication with the beam waist radius w_0. Both of these are "derivative parameters" obtained during the creation of a laser dictionary (see Sec. 2.2).

The laser may be rotated about the $\hat{\mathbf{z}}$ axis by an *azimuthal angle* θ_{xy}, which can be specified by laser['xy_angle'] (in degrees). The laser may also be offset along the (longitudinal) laser axis $\hat{\mathbf{L}}_x$ by a displacement δ_{long} (laser['long_offset']) and along the transverse axis $\hat{\mathbf{L}}_y$ by a displacement δ_{trans} (laser['trans_offset']). Both of these displacements should be given in Å.

The relationship between the `S` and `L` coordinates is given by

$$\begin{cases} L_x = \frac{\lambda_e f_{OL}}{z_R}\left(s_x \cos\theta_{xy} + s_y \sin\theta_{xy} - \frac{\delta_{long}}{\lambda_e f_{OL}}\right) \\ L_y = \frac{\lambda_e f_{OL}}{w_0}\left(-s_x \sin\theta_{xy} + s_y \cos\theta_{xy} - \frac{\delta_{trans}}{\lambda_e f_{OL}}\right) \end{cases} \tag{17}$$

where f_{OL} is the focal length of the objective.

The peak phase shift η_0 is calculated according to [1]

$$\eta_0 = \sqrt{\frac{2}{\pi^3}}\alpha\frac{P\lambda_e\lambda_L N_A}{\hbar c^2} \tag{18}$$

where P is the circulating power, α is the fine structure constant, and λ_e and λ_L are the wavelengths of the electrons and photons, respectively. This value can be calculated via `get_eta0_from_power` *or* it can be specified directly in the `laser` dictionary via `laser['peak_phase_deg']` (in degrees), in which case the laser power will be calculated from Eq. 18 via `get_eta0_from_peak_phase_deg` and `get_laser_power_from_eta0`.

The gradient of η will come in handy in Sec. 2.4.3. We calculate this gradient via

$$\begin{aligned} \frac{\partial\eta(\mathbf{s})}{\partial\mathbf{s}} &= \frac{\partial\eta}{\partial s_x}\hat{\mathbf{s}}_x + \frac{\partial\eta}{\partial s_y}\hat{\mathbf{s}}_y & (19) \\ &= \left(\frac{\partial\eta}{\partial L_x}\frac{\partial L_x}{\partial s_x} + \frac{\partial\eta}{\partial L_y}\frac{\partial L_y}{\partial s_x}\right)\hat{\mathbf{s}}_x + \left(\frac{\partial\eta}{\partial L_x}\frac{\partial L_x}{\partial s_y} + \frac{\partial\eta}{\partial L_y}\frac{\partial L_y}{\partial s_y}\right)\hat{\mathbf{s}}_y & (20) \end{aligned}$$

where

$$\begin{aligned} \frac{\partial\eta}{\partial L_x} =& \frac{\eta_0}{2N_A^2(1+L_x^2)^{11/4}}\exp\left(-\frac{2L_y^2}{1+L_x^2}\right)\Bigg\{N_A^2 L_x(1+L_x^2)^{1/4}(4L_y^2-1-L_x^2) \\ &+\exp\left(-\frac{2\theta_t^2(1+L_x^2)}{N_A^2}\right)\Bigg[L_x\cos\left(\phi_L - \frac{4L_x}{N_A^2} - \frac{2L_xL_y^2}{1+L_x^2} + \frac{3}{2}\arctan(L_x)\right) \\ &\times\left[-4\theta_t^2(1+L_x^2)^2 + N_A^2\left(4L_y^2 - \frac{3}{2}(1+L_x^2)\right) + \beta_e^2\cos^2\theta_p\left[8\theta_t^2(1+L_x^2)^2 + N_A^2\left(3(1+L_x^2) - 8L_y^2\right)\right]\right] \\ &+\left[-2(1+L_x^2)^2 + N_A^2\left(\frac{3}{4} - L_y^2 + L_x^2\left(\frac{3}{4} + L_y^2\right)\right)\right]\left[(2\beta_e\cos\theta_p)^2 - 2\right] \\ &\times\sin\left(\phi_L - \frac{4L_x}{N_A^2} - \frac{2L_xL_y^2}{1+L_x^2} + \frac{3}{2}\arctan(L_x)\right)\Bigg]\Bigg\} & (21) \\ \frac{\partial\eta}{\partial L_y} =& \frac{2\eta_0 L_y}{(1+L_x^2)^{7/4}}\exp\left(-\frac{2\theta_t^2(1+L_x^2)}{N_A^2} - \frac{2L_y^2}{1+L_x^2}\right)\Bigg\{\left[\beta_e^2(1+\cos(2\theta_p)) - 1\right] \\ &\times\left[\cos\left(\phi_L - \frac{4L_x}{N_A^2} - \frac{2L_xL_y^2}{1+L_x^2} + \frac{3}{2}\arctan(L_x)\right) - L_x\sin\left(\phi_L - \frac{4L_x}{N_A^2} - \frac{2L_xL_y^2}{1+L_x^2} + \frac{3}{2}\arctan(L_x)\right)\right] \\ &-\exp\left(\frac{2\theta_t^2(1+L_x^2)}{N_A^2}\right)(1+L_x^2)^{1/4}\Bigg\} & (22) \end{aligned}$$

I know, right?

The calculation of this gradient (Eq. 19) is performed by `get_laser_phase_gradient`.

2.4.2.2 Zernike phase plate

Although, clearly, a laser cannot produce a "true" Zernike phase plate, this phase profile may be used in place of the LPP phase profile by setting `laser['true_zernike_mode']=True`. The resulting phase profile is given by

$$\eta(\mathbf{s}) = \begin{cases} \eta_0 & \text{if } s < s_Z \\ 0 & \text{else} \end{cases} \tag{23}$$

where s_Z is the *cut-on frequency*. The cut-on frequency is set by `laser['zernike_cut_on']`. The phase is calculated with `get_laser_phase`, just as it would be for the LPP, and the peak phase shift can be specified directly via `laser['peak_phase_deg']`. If the peak phase shift is not specified, a $\pi/2$ phase shift will be assumed.

The gradient of the Zernike phase plate, or for that matter a general phase function η, can be calculated numerically with `get_numerical_phase_plate_gradient`, which uses `numpy.gradient`. In general, the analytical form of the gradient ought to be used when available (e.g. Eq. 19) to avoid issues with sampling in the back focal plane.

2.4.3 Partial coherence

The effect of partial coherence on the transfer function is derived by considering (incoherently) the contributions of two effects to reducing the coherence: (i) a cone of illumination angles with non-negligible size and (ii) chromatic aberration of the objective leading to a small spread of defocus values. The transfer function can then be described by

$$H(\mathbf{s}; f_0) = \int \exp\left[-i\chi(\mathbf{s}+\mathbf{s}_\beta; f_0+\delta_f)\right] p(\mathbf{s}_\beta) p(\delta_f) d\delta_f d^2 s_\beta \tag{24}$$

where $p(\mathbf{s}_\beta)$ and $p(\delta_f)$ are probability distributions. Tilt of the illumination in the object plane by an angle $\boldsymbol{\beta} = (\beta_x, \beta_y)$ with respect to the optical axis (of the microscope) $\hat{\mathbf{z}}$ corresponds to a translation of the unscattered beam by the displacement $\mathbf{s}_\beta = \boldsymbol{\beta}/\lambda_e$, and a fluctuation in the focus position due to chromatic aberration corresponds to a change in the *mean* focus by a distance δ_f. It is assumed that astigmatism remains unaffected, since it's a higher-order aberration than defocus.

If we assume Gaussian distributions for $\boldsymbol{\beta}$ and δ_f:

$$\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \sigma_\beta^2 \mathbf{I}) \implies \mathbf{s}_\beta \sim \mathcal{N}(\mathbf{0}, \sigma_s^2 \mathbf{I}) \tag{25}$$

$$\delta_f \sim \mathcal{N}(0, \sigma_f^2) \tag{26}$$

(where $\mathbf{I}$ is the identity matrix, i.e. the covariance matrix of $\boldsymbol{\beta}$ is diagonal, and $\sigma_s = \sigma_\beta/\lambda_e$) and we expand χ according to

$$\chi(\mathbf{s}+\mathbf{s}_\beta; f_0+\delta_f) = \chi(\mathbf{s}; f_0) + \mathbf{s}_\beta \cdot \left(\frac{\partial\chi}{\partial\mathbf{s}}\right) + \delta_f \left(\frac{\partial\chi}{\partial f_0}\right) + \delta_f \mathbf{s}_\beta \cdot \left(\frac{\partial^2\chi}{\partial\mathbf{s}\partial f_0}\right) + \cdots \tag{27}$$

we can perform the integration in Eq. 24 analytically. Making the assumption that $\eta(\mathbf{s}; f_0) = \eta(\mathbf{s})$, we arrive at the expression

$$\begin{aligned} H(\mathbf{s}; f_0) = & \frac{1}{\sqrt{1+\sigma_s^2\sigma_f^2|\mathbf{V}_3|^2}} \exp\left\{-\frac{\sigma_s^2|\mathbf{V}_1|^2}{2} - \frac{\sigma_f^2\left[V_2^2 - \left(\sigma_s^2\mathbf{V}_1\cdot\mathbf{V}_3\right)^2\right]}{2\left(1+\sigma_s^2\sigma_f^2|\mathbf{V}_3|^2\right)}\right\} \\ & \times \exp\left\{i\left[-\chi(\mathbf{s}, f_0) + \frac{\sigma_s^2\sigma_f^2 V_2 \mathbf{V}_1\cdot\mathbf{V}_3}{1+\sigma_s^2\sigma_f^2|\mathbf{V}_3|^2}\right]\right\} \end{aligned} \tag{28}$$

where

$$\mathbf{V}_1 \triangleq \frac{\partial\chi(\mathbf{s}; f_0)}{\partial\mathbf{s}} = \frac{\partial\gamma(\mathbf{s}; f_0)}{\partial\mathbf{s}} + \frac{\partial\eta(\mathbf{s})}{\partial\mathbf{s}} \tag{29}$$

$$V_2 \triangleq \frac{\partial\chi(\mathbf{s}; f_0)}{\partial f_0} = \frac{\partial\gamma(\mathbf{s}; f_0)}{\partial f_0} = -\pi\lambda_e s^2 \tag{30}$$

$$\mathbf{V}_3 \triangleq \frac{\partial^2\chi(\mathbf{s}, f_0)}{\partial\mathbf{s}\partial f_0} = \frac{\partial^2\gamma(\mathbf{s}, f_0)}{\partial\mathbf{s}\partial f_0} = -2\pi\lambda_e\mathbf{s} \tag{31}$$

Writing out the transfer function as in Eq. 6, we have

$$H(\mathbf{s}; f_0) = A(\mathbf{s}; f_0) \exp\left[i\xi(\mathbf{s}; f_0)\right] \tag{32}$$

$$A(\mathbf{s}; f_0) = \frac{1}{\sqrt{1 + 4\pi^2\sigma_s^2\sigma_f^2\lambda^2 s^2}} \exp\left\{ \frac{\sigma_f^2\pi^2\lambda^2\left[s^4 + 4\sigma_s^4\left(\mathbf{s}\cdot\left(\frac{\partial\gamma(\mathbf{s};f_0)}{\partial\mathbf{s}} + \frac{\partial\eta(\mathbf{s})}{\partial\mathbf{s}}\right)\right)^2\right]}{2(1 + 4\pi^2\sigma_s^2\sigma_f^2\lambda^2 s^2)} - \frac{\sigma_s^2}{2}\left|\frac{\partial\gamma(\mathbf{s}; f_0)}{\partial\mathbf{s}} + \frac{\partial\eta(\mathbf{s})}{\partial\mathbf{s}}\right|^2 \right\} \tag{33}$$

$$\xi(\mathbf{s}; f_0) = -\left[\gamma(\mathbf{s}; f_0) + \eta(\mathbf{s})\right] + \frac{2\pi^2\sigma_s^2\sigma_f^2\lambda^2 s^2}{1 + 4\pi^2\sigma_s^2\sigma_f^2\lambda^2 s^2}\mathbf{s}\cdot\left[\frac{\partial\gamma(\mathbf{s}; f_0)}{\partial\mathbf{s}} + \frac{\partial\eta(\mathbf{s})}{\partial\mathbf{s}}\right] \tag{34}$$

We can then plug Eq. 14 and an expression for the gradient of η (e.g. Eq. 19) into the above to calculate the partially-coherent transfer function. The components of H in Eqs. 33 and 34 are determined by `get_transfer_func_mag_and_phase`.

We can also observe that if the cross term $\mathbf{V}_3$ is neglected, Eq. 28 reduces to

$$H(\mathbf{s}; f_0) = \exp\left(-\frac{\sigma_s^2|\mathbf{V}_1|^2}{2} - \frac{\sigma_f^2 V_2^2}{2}\right)\exp\left[-i\chi(\mathbf{s}, f_0)\right] \tag{35}$$

This form of the transfer function can be used by setting `scope['enable_cross_term']=False`. We can see the spatial and temporal coherence envelopes, corresponding to the first and second terms in the exponential decay, respectively. We will discuss these two terms next to get some understanding of σ_s and σ_f.

2.4.3.1 Temporal coherence and σ_f

The second term is readily written out via Eq. 30:

$$E_{TC}(\mathbf{s}) = \exp\left(-\frac{\sigma_f^2 V_2^2}{2}\right) \tag{36}$$

$$= \exp\left(-\frac{1}{2}\sigma_f^2\pi^2\lambda_e^2 s^4\right) \tag{37}$$

and with reference to an abundance of sources in the literature we call σ_f the *defocus spread*, which is determined by the fluctuations of the electron energies E_0, accelerating voltages V_0, and lens currents I_0, scaled by the chromatic aberration coefficient, C_c. According to Bart Buijsse (ThermoFisher/FEI), this is given by

$$\sigma_f = C_c\sqrt{\left(\epsilon\frac{\sigma_{V_0}}{V_0}\right)^2 + \left(\epsilon\frac{\sigma_{E_0}}{E_0}\right)^2 + \left(2\frac{\sigma_{I_0}}{I_0}\right)^2} \tag{38}$$

$$\epsilon = \frac{1 + \frac{E_0}{511\text{ keV}}}{1 + \frac{E_0}{2\cdot 511\text{ keV}}} \approx 1.227 \text{ (at 300 keV)} \tag{39}$$

$$\Delta E_0 = 2\sqrt{2\ln 2}\,\sigma_{E_0} \approx 2.355\sigma_{E_0} \tag{40}$$

$$\frac{\sigma_{I_0}}{I_0} = 0.1\text{ ppm} \tag{41}$$

$$\frac{\sigma_{V_0}}{V_0} = 0.07\text{ ppm} \tag{42}$$

The "energy spread" of electron sources is reported in FWHM, i.e. as ΔE_0. There is a relativistic correction term ϵ that must be included to be consistent with ThermoFisher's specs on C_c. Note that textbooks such as Spence (Eq. 4.9 of [3]) leave this relativistic factor out!! The defocus spread term is calculated using `get_defocus_spread`. In the code, these terms are determined according to

C_c :	`scope['C_c']`
E_0 :	`scope['HT']`
σ_{E_0} :	`scope['energy_spread_std']`
$\frac{\sigma_{V_0}}{V_0}$:	`scope['accel_voltage_std']`
$\frac{\sigma_{I_0}}{I_0}$:	`scope['OL_current_std']`

Note that $V_0 = E_0 = V_{HT}$. Importantly, `scope['C_c']` should be specified in Å rather than mm. Also, `scope['accel_voltage_std']` and `scope['OL_current_std']` should be specified as a fraction. It is expected that the first term in the square root, due to electron energy spread, will dominate this expression.

Note: Various sources in the literature will sometimes report Eq. 37 written with a different factor than $-\frac{1}{2}$. This is due to the use of σ_f to denote the $1/e$ width of $p(\delta_f)$ instead of standard deviation. Since σ canonically means standard deviation, we will use it to mean standard deviation.

2.4.3.2 Spatial coherence and σ_s

The spatial coherence envelope corresponds to the first term in Eq. 35:

$$\exp\left(-\frac{\sigma_s^2 |\mathbf{V}_1|^2}{2}\right) = \exp\left(-\frac{\sigma_s^2}{2}\left|\frac{\partial\gamma(\mathbf{s};f_0)}{\partial\mathbf{s}} + \frac{\partial\eta(\mathbf{s})}{\partial\mathbf{s}}\right|^2\right) \tag{43}$$

If we neglect the phase plate, or if there simply isn't one, then we can recover the spatial coherence envelope available in the literature via Eq. 14. We will have to make the further approximation to Eq. 14 that $f_a = 0$, such that

$$\frac{\partial\gamma(\mathbf{s};f_0)}{\partial\mathbf{s}} = 2\pi(C_s\lambda_e^3 s_\rho^3 - f_0\lambda_e s_\rho)\left(\cos s_\varphi \hat{\mathbf{s}}_x + \sin s_\varphi \hat{\mathbf{s}}_x\right) \tag{44}$$

$$\implies \left|\frac{\partial\gamma(\mathbf{s};f_0)}{\partial\mathbf{s}}\right|^2 = 4\pi^2\lambda_e^2 s^2\left(C_s\lambda_e^2 s^2 - f_0\right)^2 \tag{45}$$

once again making use of the notation $s_\rho = s$. The simplified spatial coherence envelope thus has the form

$$E_{SC}(\mathbf{s}) = \exp\left[-2\sigma_s^2\pi^2\lambda_e^2 s^2\left(C_s\lambda_e^2 s^2 - f_0\right)^2\right] \tag{46}$$

$$= \exp\left[-2\sigma_\beta^2\pi^2 s^2\left(C_s\lambda_e^2 s^2 - f_0\right)^2\right] \tag{47}$$

where we have used $\sigma_\beta = \lambda_e\sigma_s$. This is the form of the spatial coherence envelope that is widely available in the literature. In the code, this simpler form can *almost* be obtained by setting `scope['enable_cross_term']=False` and `scope['enable_phase_plate_coherence']=False`. The former sets the cross term (Eq. 31) to zero while the latter sets the gradient of η to zero in Eq. 43. What remains is to set $f_a = 0$, which you can do via `img['astig_z']=0`, but then you won't have astigmatism.

In order to calculate the spatial coherence envelope, the user must specify σ_β, the *beam semiangle*. This is available in the `scope` dictionary:

σ_β :	`scope['beam_semiangle']`

Note: Various sources in the literature will sometimes report Eq. 47 written with a different multiplicative factor. This is due to the use of σ_s to denote the $1/e$ width of $p(\mathbf{s}_\beta)$ instead of standard deviation. Since σ canonically means standard deviation, we will use it to mean standard deviation.

2.4.4 The contrast transfer function

The contrast transfer function $\mathrm{CTF}(\mathbf{s})$ can be computed using `get_transfer_func` by setting the argument `return_ctf=True`. To work out the math, we consider a weak phase object with a *transmission function*

$$t(\mathbf{x}) = 1 + i\phi(\mathbf{x}) \tag{48}$$

We then have the *exit wave* $\psi_{op}(\mathbf{x}) = t(\mathbf{x})$ such that Eq. 1 can be written via Eq. 6 as

$$\begin{aligned} \psi_{bfp}(\mathbf{s}) &= \mathcal{F}\left[1 + i\phi(\mathbf{x})\right](\mathbf{s}) \cdot A(\mathbf{s})e^{i\xi(\mathbf{s})} && (49)\\ &= \left(\delta(\mathbf{s}) + i\mathcal{F}\left[\phi(\mathbf{x})\right](\mathbf{s})\right) \cdot A(\mathbf{s})e^{i\xi(\mathbf{s})} && (50) \end{aligned}$$

where $\delta(\mathbf{s})$ is a delta function. Then, via Eqs. 2 and 3, we have in the image plane

$$\begin{aligned} \psi_{ip}(\mathbf{x}) &= \mathcal{F}^{-1}\left[\psi_{bfp}(\mathbf{s})\right](\mathbf{x}) && (51)\\ &= A(\mathbf{0})e^{i\xi(\mathbf{0})} + i\phi(\mathbf{x}) * \mathcal{F}^{-1}\left[A(\mathbf{s})e^{i\xi(\mathbf{s})}\right](\mathbf{x}) && (52)\\ I_{ip}(\mathbf{x}) &= \left|\psi_{ip}(\mathbf{x})\right|^2 && (53)\\ &= \left|A(\mathbf{0})e^{i\xi(\mathbf{0})} + i\phi(\mathbf{x}) * \mathcal{F}^{-1}\left[A(\mathbf{s})e^{i\xi(\mathbf{s})}\right](\mathbf{x})\right|^2 && (54)\\ &\approx A^2(\mathbf{0}) + iA(\mathbf{0})e^{-i\xi(\mathbf{0})}\phi(\mathbf{x}) * \mathcal{F}^{-1}\left[A(\mathbf{s})e^{i\xi(\mathbf{s})}\right](\mathbf{x}) - iA(\mathbf{0})e^{i\xi(\mathbf{0})}\phi(\mathbf{x}) * \mathcal{F}^{-1}\left[A(\mathbf{s})e^{i\xi(\mathbf{s})}\right]^*(\mathbf{x}) && (55) \end{aligned}$$

where we have dropped the term quadratic in $\phi(\mathbf{x})$ in the last line. The contrast transfer function is obtained by Fourier transforming $I_{ip}(\mathbf{x})$:

$$\begin{aligned} \mathcal{F}[I_{ip}(\mathbf{x})](\mathbf{s}) &= A^2(\mathbf{0})\delta(\mathbf{s}) + iA(\mathbf{0})e^{-i\xi(\mathbf{0})}\mathcal{F}[\phi(\mathbf{x})](\mathbf{s}) \cdot A(\mathbf{s})e^{i\xi(\mathbf{s})} - iA(\mathbf{0})e^{i\xi(\mathbf{0})}\mathcal{F}[\phi(\mathbf{x})](\mathbf{s}) \cdot A(-\mathbf{s})e^{-i\xi(-\mathbf{s})} && (56)\\ &= A^2(\mathbf{0})\delta(\mathbf{s}) + \mathcal{F}[\phi(\mathbf{x})](\mathbf{s}) \cdot \left[iA(\mathbf{0})e^{-i\xi(\mathbf{0})}A(\mathbf{s})e^{i\xi(\mathbf{s})} - iA(\mathbf{0})e^{i\xi(\mathbf{0})}A(-\mathbf{s})e^{-i\xi(-\mathbf{s})}\right] && (57)\\ &= A^2(\mathbf{0})\delta(\mathbf{s}) - 2\mathcal{F}\left[\phi(\mathbf{x})\right](\mathbf{s}) \cdot \mathrm{CTF}(\mathbf{s}) && (58) \end{aligned}$$

where

$$\begin{aligned} \mathrm{CTF}(\mathbf{s}) &\triangleq -\frac{iA(\mathbf{0})}{2}\left[e^{-i\xi(\mathbf{0})}A(\mathbf{s})e^{i\xi(\mathbf{s})} - e^{i\xi(\mathbf{0})}A(-\mathbf{s})e^{-i\xi(-\mathbf{s})}\right] && (59)\\ &= -\frac{iA(\mathbf{0})}{2}\left\{A(\mathbf{s})e^{i[\xi(\mathbf{s})-\xi(\mathbf{0})]} - A(-\mathbf{s})e^{-i[\xi(-\mathbf{s})-\xi(\mathbf{0})]}\right\} && (60) \end{aligned}$$

In `get_transfer_func`, the contrast transfer function $\mathrm{CTF}(\mathbf{s})$ is determined from Eq. 60 by first calculating $A(\mathbf{s})$ and $\xi(\mathbf{s})$ via Eqs. 33 and 34. Those calculations are implemented in `get_transfer_func_mag_and_phase`.

The amplitude transfer function is also determined by `get_transfer_func` if the setting `return_ctf=True` is used. The definition of $\mathrm{ATF}(\mathbf{s})$ follows a similar derivation to the above (with a small amplitude term in $t(\mathbf{x})$). It is defined as

$$\mathrm{ATF}(\mathbf{s}) \triangleq \frac{A(\mathbf{0})}{2}\left\{A(\mathbf{s})e^{i[\xi(\mathbf{s})-\xi(\mathbf{0})]} + A(-\mathbf{s})e^{-i[\xi(-\mathbf{s})-\xi(\mathbf{0})]}\right\} \tag{61}$$

Note: In the special case

$$\begin{aligned} A(\mathbf{s}) &= A(-\mathbf{s}) && (62)\\ \xi(\mathbf{s}) &= \xi(-\mathbf{s}) && (63)\\ A(\mathbf{0}) &= 1 && (64)\\ \xi(\mathbf{0}) &= -\eta_0 && (65) \end{aligned}$$

the CTF as defined in Eq. 60 reduces to

$$\mathrm{CTF}(\mathbf{s}) = A(\mathbf{s})\sin\left[\xi(\mathbf{s}) + \eta_0\right] \tag{66}$$

and the ATF as defined in Eq. 61 reduces to

$$\mathrm{ATF}(\mathbf{s}) = A(\mathbf{s})\cos\left[\xi(\mathbf{s}) + \eta_0\right] \tag{67}$$

2.5 The sample potential

The sample potential is considered in the *Independent Atom Model* (IAM), wherein the potential due to N atoms is simply the sum of their potentials. The potential is calculated for a collection of atoms by `get_potential_from_atom_data`.

2.5.1 Importing the atoms

Before the potential can be calculated, the `atom_data` must be created or imported. To read it from a `*.pdb` or `*.cif` file, use `read_structure`. The output is a dictionary containing the atomic XYZ coordinates `'coords'` in a $N \times 3$ array (in Å), the `'id'` number (atomic number) of each element, and the element's text symbol, `'element'`. To specify the file location, set `sample['struct_file_loc']`. A few proteins are hardcoded in the function `get_struct_file_loc` and can instead be specified by alternate names, `sample['sample_name']`. The protein can also be rotated or translated by specifying Euler angles (in degrees) with `sample['xyz_angles_deg']` or a 3D displacement (in Å) with `sample['xyz_loc']`. Note that in the guts of the code there are (x, y, z) coordinates and (r, c, z) coordinates, the latter standing for row, column, z. That's because of the way `matplotlib` renders images of `numpy` arrays.

There are a couple of options which provide alternatives to importing a protein. A single atom can be simulated with `sample['name']='atom_X'` where `X` is the atom's `element` string. A random collection of carbon atoms in a volume can be simulated with `sample['name']='random_carbon']`. And a few test objects can be simulated with different `sample['name']`: `'test_plane'`, a 2D Gaussian noise with standard deviation `sample['test_phase']`; `'test_circle'`, a circle with a diameter `sample['circle_diam']` (in Å); `'test_circle_array'`, an array of such circles.

2.5.2 Atomic potentials

The potential due to an atom is calculated by `get_atomic_potential`. This function makes a grid of size `n_grid_pix` (e.g. 11) and supersamples it by a factor `ss_factor` (e.g. 8). The atom's potential is evaluated on this supersampled grid (e.g. 88×88) according to Eq. C.20 in [4].

$$V_a(x, y, z) = 2\pi^2 a_0 e \sum_{i=1}^{3} \frac{a_i}{r} \exp\left(-2\pi r\sqrt{b_i}\right) + 2\pi^{5/2} a_0 e \sum_{i=1}^{3} c_i d_i^{-3/2} \exp\left(\frac{-\pi^2 r^2}{d_i}\right) \tag{68}$$

where $a_0 = \hbar^2/m_0 e^2$ is the Bohr radius, $r^2 = x^2 + y^2 + z^2$ and the coefficients $\{a_i, b_i, c_i, d_i\}$ are different for each element (and looked up in `fparams.csv`).

2.5.3 The solvent potential

The solvent potential is implemented using the Shang-Sigworth continuum model [5]. This model is implemented in `get_shang_sigworth_solvent`. The calculation of the potential happens in two steps which will be explained further in the next sections:

1. The calculation of initial solvent density based on distance to nearest atom
2. The calculation of a mask that encodes which positions are inside/outside the protein.

Once both of these calculations are done, they are multiplied to give the final solvent density. The purpose of the mask is to ensure that empty pockets inside the protein do not have any solvent in them, to better approximate reality. The final solvent potential can be calculated by taking the final density and multiplying it by `3.6*z_pixel_size`. Then this final solvent potential is added to the atomic potential to give the final potential used for exit wave calculation.

2.5.4 Calculating the initial solvent density

The initial step in calculating the solvent is calculating the estimated density based only on the distance to the nearest atom (technically the distance to the nearest Van der Waals radius of an atom). This initial solvent density can be found by evaluating this function (t for type of atom [i.e. polar or non-polar]):

$$F_t(r) = \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left[\frac{r - r_{1t}}{\sqrt{2}\sigma_1}\right] + a_{2t}\exp\left[-\frac{(r-r_{2t})^2}{2\sigma_{2t}^2}\right] + a_{3t}\exp\left[-\frac{(r-r_{3t})^2}{2\sigma_{3t}^2}\right]. \tag{69}$$

The constants in this function are listed in the table below (and hardcoded in `get_shang_sigworth_density` which computes the above equation):

Constant	Polar	Non-Polar
a_2	0.2	0.15
a_3	-0.15	-0.12
r_1	0.5	1
r_2	1.7	2.2
r_3	1.7	3.6
σ_1	1	1
σ_2	1.77	1.77
σ_3	1.06	0.85

This function is evaluated at every sample point in the potential, with the value equal to the distance to the nearest atom plus its van der Waals radius. The result of this computation is the initial solvent potential.

2.5.5 Calculating the solvent mask

The calculation of the solvent mask is the most computationally intensive part of the entire calculation of the potential. This calculation is performed as such:

1. a binary mask the size of the potential is created and all values are set to 1
2. all points in the mask within a distance of van der Waals radius `vdw_rad` plus `r_probe` are set to 0
3. a list of all "boundary points" is made by searching through the entire mask to find all points with a neighbor that has a different value than the point in question
4. all points within `r_shrink = vdw_rad + distance_to_nearest_nucleus` are set back to 1

The last step in this calculation is the most computationally intensive step, but it is crucial for an accurate solvent density model. This "flood-and-fill" algorithm ensures that pockets inside the protein structure do not contain any solvent, giving a more accurate approximation.

2.5.6 Thermal vibration

The function `apply_thermal_gaussian` applies a thermal vibration to the potential by convolving it with a Gaussian filter of standard deviation `sample['vibration']` (in Å).

2.6 The exit wave, $\psi_{op}(\mathbf{x})$

2.6.1 The multislice method

The exit wave is computed by a multislice calculation which begins with an incident wavefunction `psi_in` such as a plane wave, `psi_in=1`. If something other than a constant is desired, it should be a 2D array matching the lateral dimensions of the potential.

The propagation from the n^{th} to the $(n+1)^{\mathrm{th}}$ slice is described by

$$\psi_{n+1}(\mathbf{x}) = \mathcal{F}^{-1}\left[\mathcal{F}\left[\psi_n(\mathbf{x})\exp\left[i\sigma_e v(\mathbf{x};\zeta_{n+1})\right]\right](\mathbf{s})P(\mathbf{s};\Delta)\right](\mathbf{x}), \tag{70}$$

where $P(\mathbf{s};\Delta)$ is the Fourier transform of the Fresnel kernel,

$$P(\mathbf{s};\Delta) = \mathcal{F}\left[\frac{1}{i\lambda_e\Delta}\exp\left(\frac{i\pi\left|\mathbf{x}\right|^2}{\lambda_e\Delta}\right)\right](\mathbf{s}) \tag{71}$$

$$= \exp\left[-i\pi\lambda_e\Delta s^2\right], \tag{72}$$

with $\mathbf{s}$ denoting the (non-angular) spatial frequency and $s = |\mathbf{s}|$. The interaction parameter σ_e is the scaling factor between sample potential and phase, given by

$$\sigma_e = \frac{2\pi}{\lambda_e V}\left(\frac{m_0c^2 + eV}{2m_0c^2 + eV}\right) \tag{73}$$

and calculated when initializing a `scope` as part of a call to `get_deriv_params`. Here, m_0c^2 is the electron's rest energy, V is the accelerating voltage, e is the electron's charge. The thickness of each slice is Δ, which is set in the code as `sample['slice_thickness']/axial_pixel_size`. The default value of `axial_pixel_size` is 1 Å, but it can be changed in `do_multislice` or its parent `get_exit_wave_op`. For a sample with N slices, the wavefunction that emerges might be called $\psi_N(\mathbf{x})$. Since samples tend to have a mean z position of zero, one more Fresnel propagation is applied:

$$\psi_{ew}(\mathbf{x}) = \mathcal{F}^{-1}\left[\mathcal{F}\left[\psi_N(\mathbf{x})\right](\mathbf{s})P\left(\mathbf{s};\frac{(N-1)\Delta}{2}\right)\right](\mathbf{x}). \tag{74}$$

A transfer function having a defocus of 0 will then produce an in-focus image of the $z = 0$ plane. In order to check for consistency, as suggested in [4], the integral of the wavefunction is computed at each step and its mean and standard deviation are displayed after an exit wave calculation is completed.

3 Running the thing

3.1 Requirements

The only package that is really needed apart from regular `scipy`, `numpy`, `matplotlib` type stuff is Biopython, which handles the importing of files from the PDB. Additionally, `fparams.csv` is needed for the calculation of atomic potentials, and of course the PDB files themselves are needed (`*.pdb` or `*.cif` for larger files).

3.2 Inputs

In general, distances are in Å, which means mm usually appear as 10^7. The core components are

- `camera=tem.make_camera()`: settings related to the camera, e.g. pixel size and final dimensions of the image
- `scope=tem.make_scope()`: settings related to the TEM, e.g. aberrations, accelerating voltage, coherence parameters . . .
- `laser=tem.make_laser()`: settings related to the LPP, e.g. laser cavity settings, alignment in the microscope, . . .
- `sample=tem.make_sample()`: settings related to the sample, e.g. specimen type, ice thickness, multislice parameters . . .
- `img=tem.specify_img_features()`: settings related to an image, i.e. defocus and astigmatism parameters.

The documentation for these functions is pretty good (`help(tem.make_camera)` for example).

3.3 Example

Various plotting functions and high-level functions such as `do_microscopy` exist in the code but it may be more useful to use lower-level functions. Both are shown in the example below.

```
# example26.py
import tem26 as tem
import numpy as np
from matplotlib import pyplot as plt

laser = tem.make_laser(peak_phase_deg=90, NA=0.04)
camera = tem.make_camera(rows=2048, cols=2048, pixel_size=1)
scope = tem.make_scope(HT=300e3, Cc=7.2e7, Cs=4.8e7, focal_length=20e7,
    energy_spread_std=tem.get_sigma_from_fwhm(0.8),
    coherence_envelopes_on=False, dose=100)
img = tem.specify_img_features(mean_z=15000, astig_z=0)

# get transfer function
(scope, laser) = tem.get_transfer_func(img, scope, camera, laser,
    ctf_mode=False, return_ctf_components=False)

# alternative way to get transfer function
(A, xi, _, _) = tem.get_transfer_func(img, scope, camera, laser,
    ctf_mode=True, return_ctf_components=True)
scope['H_bfp'] = A*np.exp(1j*xi)
laser = tem.get_laser_phase(laser, scope, camera, tem.const)

# samples
test = tem.make_sample(name='test_plane', test_phase=0.1) # gaussian noise
protein = tem.make_sample(name='myoglobin', thickness=100, solvated=True,
    slice_thickness=10) # myoglobin

fig, ax = plt.subplots(2, 2, figsize=(10, 10))
for ind, sample in enumerate([test, protein]):
    psi_in = 1 # plane wave illumination
    ew = tem.get_exit_wave_op(sample, psi_in, scope, camera) # exit wave
    psi = tem.TEM(ew['psi'], laser, scope, camera) # image wavefunction
    pdf = tem.calc_image_pdf(psi) # image probability density
    im = tem.pdf_to_image(pdf, scope['dose'], camera['pixel_size'],
        noise_flag=True) # noisy image
    asd = tem.calc_amplitude_spectrum(im) # amplitude spectrum

    ax[ind, 0].imshow(im[camera['rows'] - 50:camera['rows'] + 50,
                         camera['cols'] - 50:camera['cols'] + 50])
    ax[ind, 0].set_title('im:␣{}␣(zoom)'.format(sample['name']))
    ax[ind, 1].imshow(np.log10(asd))
    ax[ind, 1].set_title('asd:␣{}'.format(sample['name']))

# built in functions and plotters
results = tem.do_microscopy(img, protein, laser, camera, scope)
tem.plot_results(results)
tem.plot_bfp(laser, camera, scope)
tem.plot_lpp(laser, camera, scope)
tem.plot_ctf(img, laser, camera, scope)
```

References

[1] H. Müller et al., New J. Phys. 2010

[2] L. Reimer, H. Kohl, *Transmission Electron Microscopy: Physics of Image Formation*, 2008

[3] J.C.H. Spence, *High-Resolution Electron Microscopy*, 2003

[4] E.J. Kirkland, *Advanced Computing in Electron Microscopy*, 2020

[5] Z. Shang, F.J. Sigworth, J. Struct. Biol. 2012