Pythonreflectivity v3.0

M. Zwiebler

February 6, 2018

Preliminary remarks

Installation:
Unpack the Pythonreflectivity folder. Open the folder in a console window. Install with "python setup.py install". Under Windows there troubles may occur that involve the C compiler and vcvarsall.bat, but those can usually be solved with Google.
The Pythonreflectivity package geometry is inspired by *ReMagX* [1], which was also used to cross-check the reflectivity results. Note that the conventions are different, because most of the math was derived based on Abdulhalim's paper [2]. *ReMagX* uses a coordinate system that is mirrored at the scattering plane. To go from one sign convention to the other, just transpose $\hat{\hat{\chi}}$. Run the script "Examples.py" for calculation examples.

Geometry

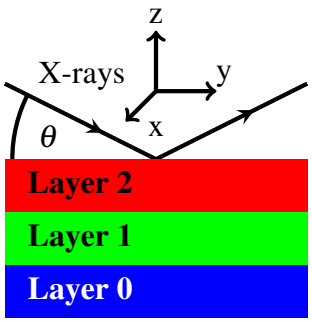

Figure 1: Reflectivity geometry

List of Functions and commands

Pythonreflectivity.Generate_structure(N, (MLstructure))

This command generates a structure on which the reflectivity can be calculated.
Input:

- Number of layer types N (int)
- Multilayer structure (optional), from the lowest layer to the highest. Default is "$0,1,2,3,...,N-1$". Multilayers syntax is "$0,100*(1,2,3,2),4,...$". Brackets inside brackets is not supported.
- A calculation with the multilayer structure "$0,N*(1,2),3$" will require calculation time on the order of $\log(N)$ and is therefore vastly superiour to "$0,1,2,...,1,2,3$" which needs on the order of N.

Output:
A list of Layer-type objects with length N

Layers

The most important input is the susceptibility (tensor) χ. Remember the following equations:
$1+\varepsilon=\chi$
The susceptibility +1 (or plus unity matrix) is the dielectric constant.

$n = \sqrt{\varepsilon} \approx 1 + \chi/2$ The dielectric constant is connected to the refractive index.
$\chi(E) = \sum_{i \in \text{Atoms}} \frac{4\pi\rho_i r_e f_i(0,E)}{k_0^2}$
The susceptibility is connected to the X-ray form factors $f_i(0,E)$. r_e is the Thomson scattering length. ρ_i is the molar density of atom i.
The description of X-ray scattering by means of atomic form factors is completely equivalent to a desciption based on slabs with constant susceptibility.
When a magnetic field is applied along one of the optical axes, the magnetic scattering can be described by one complex, energy-dependent magneto-optic constant g, which is to first approximation proportional to the magnetization $\hat{M}$.
Magnetic field along x:

$$\hat{\hat{\chi}} = \begin{pmatrix} \chi_{xx} & 0 & 0 \\ 0 & \chi_{yy} & ig \\ 0 & -ig & \chi_{zz} \end{pmatrix} \quad \text{for} \quad \hat{M} \parallel x$$

Magnetic field along y:

$$\hat{\hat{\chi}} = \begin{pmatrix} \chi_{xx} & 0 & -ig \\ 0 & \chi_{yy} & 0 \\ ig & 0 & \chi_{zz} \end{pmatrix} \quad \text{for} \quad \hat{M} \parallel y \tag{1}$$

Magnetic field along z:

$$\hat{\hat{\chi}} = \begin{pmatrix} \chi_{xx} & ig & 0 \\ -ig & \chi_{yy} & 0 \\ 0 & 0 & \chi_{zz} \end{pmatrix} \quad \text{for} \quad \hat{M} \parallel z$$

The Kramers-Kronig transformation holds for χ_{xx}, χ_{yy}, χ_{zz} and g.
Setting values in the program:

- Layer.setd(d) - Sets the layer thickness d. Default is 0.
- Layer.setsigma(σ) - Sets the roughness σ for the interface on top of this layer. . Default is 0.
- Layer.setmag(direction) - Sets the magnetization direction for the magneto-optical Kerr effect (MOKE). Allowed inputs are “x”, “y”, “z” and “0”. Default is “0”.
- Layer.setchi(χ) - Sets the complex dielectric tensor. Numbers/Length 1 lists set χ_{xx}, χ_{yy} and χ_{zz}. Length 3 lists set χ_{xx}, χ_{yy} and χ_{zz}. Length 4 lists set χ_{xx}, χ_{yy} and χ_{zz} and, depending on the magnetization direction, either $\chi_{yz} = -\chi_{zy}$ (if x), $\chi_{xz} = -\chi_{zx}$ (if y) or $\chi_{xy} = -\chi_{yx}$ (if z). Length 9 lists set [χ_{xx}, χ_{xy}, χ_{xz}, χ_{yx}, χ_{yy}, χ_{yz}, χ_{zx}, χ_{zy}, χ_{zz}].
- Layer.setchixx(χ_{xx}) sets χ_{xx}
- Layer.setchiyy(χ_{yy}) sets χ_{yy}
- Layer.setchizz(χ_{zz}) sets χ_{zz}
- Layer.setchig(χ_g) sets either $\chi_{zy} = -\chi_{yz}$ (if x), $\chi_{xz} = -\chi_{zx}$ (if y) or $\chi_{xy} = -\chi_{yx}$ (if z)
- The command also works equivalently with χ_{xy}, χ_{xz}, χ_{yx}, χ_{yz}, χ_{zx} and χ_{zy}.

Getting values:

- Layer.d() - returns the layer thickness d

- Layer.sigma() - returns the roughness σ for the interface on top of this layer
- Layer.mag() - returns the magnetization direction for MOKE.
- Layer.chi() - returns the complex dielectric tensor as a number, length 3 array or length 4 array depending on the input.
- Layer.chixx() returns χ_{xx}
- Layer.chiyy() returns χ_{yy}
- Layer.chizz() returns χ_{zz}
- Layer.chig() returns either $\chi_{zy} = -\chi_{yz}$ (if x), $\chi_{xz} = -\chi_{zx}$ (if y) or $\chi_{xy} = -\chi_{yx}$ (if z)
- The command also works equivalently with χ_{xy}, χ_{xz}, χ_{yx}, χ_{yz}, χ_{zx} and χ_{zy}.

Pythonreflectivity.Reflectivity(structure, angles, wavelength, (MultipleScattering), (MagneticCutoff), (Output))

Input:

- structure - A list of layers with that contains the Multilayer information generated by Pythonreflectivity.Generate_structure
- angles - array of doubles between 0° and 90°, or a single angle.
- wavelength - double
- MultipleScattering (optional) - Takes True or False. False is $\approx$ 20% faster. Default is True. Has no effect on calculations that require the full matrix.
- MagneticCutoff (optional) - double. If an off-diagonal element of χ (χ_g) fulfills $|\chi_g|$<MagneticCutoff, it is set to zero. The calculation of the magnetic reflectivity along y and z has some numerical stability issues. I think I caught them all and therefore I have set the default magnetic cutoff to 10^{-50}. If your calculated XMCD has more numerical noise than it should have, set a magnetic cutoff and contact me. Be careful, it can lead to discontinuities in the calculated reflectivity.

Output (optional) - Allowed inputs are

- "T" - All Intensities - If not magnetic, the output will be a matrix of doubles size [2, len(angles)] containing the intensity reflectivity for σ and π polarization. If magnetic the output will be a matrix of doubles size [4, len(angles)] containing the intensity for σ, π, left circular and right circular reflectivity.
- "t" - All Amplitudes - If not magnetic, the output will be a complex size [2, len(angles)] containing the amplitude reflectivity for σ and π polarization. If magnetic the output will be a complex matrix size [4, len(angles)] containing the reflectivity matrix elements $[r_{\sigma\to\sigma}, r_{\pi\to\sigma}, r_{\sigma\to\pi}, r_{\pi\to\pi}]$.
- "S" σ Intensity reflectivity - If not magnetic in y or z direction, the output will be an array of doubles containing the intensity reflectivity of σ-polarized light.

- "s" σ amplitude reflectivity - If not magnetic in y or z direction, the output will be a complex array containing the amplitude reflectivity of σ-polarized light.

- "P" π Intensity reflectivity - If not magnetic in y or z direction, the output will be an array of doubles containing the intensity reflectivity of π-polarized light.

- "p" π amplitude reflectivity - If not magnetic in y or z direction, the output will be a complex array containing the amplitude reflectivity of π-polarized light.

- The default input it "T".

Pythonreflectivity.KramersKronig(energy, absorption)

Input:

- energy - An ordered numpy array of type double and length L. **The energy points do not have to be evenly spaced, but they should be ordered**.
- absorption - An array of type double and length L with absorption data.

Output: An array of type double and length L with the Kramers-Kronig transformation of the absorption.
The idea is to determine the difference between a scaled experimental or multiplet absorption spectrum and the off-resonant form factors. This difference can then be Kramers-Kronig-transformed and added to the real part of the off-resonant form factors. Be careful with sign conventions!
If you see small differences between the results of this Kramers-Kronig implementation and others, it is most likely due to small differences in how the numerical integration is implemented.

Numerical noise

I was particularly careful with loss of significant digits during subtraction. Depending on the precise input, the calculated reflectivity should have 14-16 correct significant digits if the input (χ, d, σ, λ, θ) has all digits correct. This is important because many common fit procedures require numerical derivatives. Numerical derivatives require a subtraction and therefore you loose significant digits. Keep in mind when you calculate the XMCD, you will also do a subtraction. If the relative XMCD is very small, you will loose more significant digits. If you find a case where the number of significant digits in the reflectivity is significantly lower than expected, please contact me. If you are calculating numerical derivatives during a fit and you don't have enough significant digits left, look up Numerical differentiation on wikipedia for improvements.

Remarks on the full matrix

The program will resort to the full matrix formalism if either one layer does not obey any of the symmetries of the matrices in (1), or if there are two layers that have different magnetization directions.
The full matrix formalism is by a factor of ≈ 3 slower per layer than MOKE in the y or z direction. However, it will recognize if a layer has a symmetry that simplifies the calculations. Therefore the lower speed may not matter that much.
The formalism will have more numerical noise than MOKE or unmagnetic reflectivity calculations! I tried to reduce this as much as possible but stability always comes at a cost, which is slower execution time.

If you want to calculate changes of the reflectivity with sample rotations, the susceptibility tensor χ transforms like $\chi(\theta,\phi) = R(\theta,\phi)^T \chi(0,0) R(\theta,\phi)$. R is a rotation matrix in 3D space.
The matrix formalism has also numerical stability issues for multilayers with large N. In general, the program will get confused if the total stack is extremely thick so that deep interfaces do not contribute. In practice this can always be avoided with a suitable choice of the input heterostructure.

Approximations used

- No off-specular scattering
- The program assumes a plane wave with perfect coherence longitudinal and transversal. Finite coherence leads to off-specular scattering and can usually be taken into account with moderate effort
- No Compton effect, nonlinear optics or inelastic scattering contributions are calculated
- Calculations with roughness are based on the Névot and Croce approximation. This assumes that $\chi \ll 1$. All contributions that are quadratic in χ are neglected. In the X-ray regime this is usually acceptable. **The Névot and Croce approximation is not valid if the layer thickness is similar to the roughness. Do not use it. The results will be wrong. Seriously!**

Examples

The file "Examples.py" contains example reflectivity calculations for some reasonable situations.

Parallelization

The Pythonreflectivity package uses a single processor, but it can be parallelized from the Python side. I found the joblib module to be up to the task.

Bibliography

(1) Macke, S.; Benckiser, E.; Goering, E.; Hinkov, V. ReMagX - x-ray magnetic reflectivity tool., Max Planck -UBC centre for quantum materials., www.remagx.org.

(2) Abdulhalim, I. *Optics Communications* **1999**, *163*, 9–14.