Diffractometer Rotation Matrix

Here the rotation matrices for the diffractometer are given, based on Euler rotation matrices.

The full sample rotation is an intrinsic rotation about the following axes:

1. mu right-handed rotation about x
2. eta left-handed rotation about z'
3. chi right-handed rotation about y''
4. phi left-handed rotation about z'''

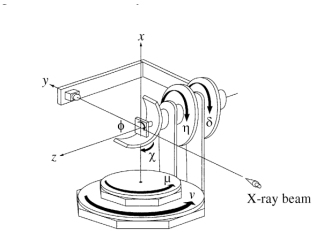
Rotations of a vector are made by pre-multiplying a column-vector:

$$R.v = \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z \end{pmatrix}$$

Contents

Useful links

- https://en.wikipedia.org/wiki/Rotation_matrix#General_rotations
- https://en.wikipedia.org/wiki/Euler_angles
- https://doi.org/10.1107/S0021889899001223
- https://diffcalc.readthedocs.io/en/latest/youmanual.html
- Matrix calculator (Wolfram Alpha)
- Matrix calculator (Maxima)



Rotation matrices

Rotation mu about the x-axis (right handed):

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\mu) & -\sin(\mu)) \\ 0 & \sin(\mu) & \cos(\mu)) \end{pmatrix}$$

Rotation Eta about the z' axis (left handed):

$$E = \begin{pmatrix} \cos(\eta) & \sin(\eta) & 0 \\ -\sin(\eta) & \cos(\eta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation Chi about the y'' axis (right handed):

$$C = \begin{pmatrix} \cos(\chi) & 0 & \sin(\chi) \\ 0 & 1 & 0 \\ -\sin(\chi) & 0 & \cos(\chi) \end{pmatrix}$$

Rotation Phi about the z''' axis (left handed):

$$P = \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

To multiply a vector, combine the matrices left-to-right (intrinsic):

$$v' = M.E.C.P.v$$

B Matrix

The B matrix can be used to transform a reflection (h,k,l) into a cartesian basis in units of inverse angstroms

$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} b_1 & b_2\cos(\beta_3) & b_3\cos(\beta_2) \\ 0 & b_2\sin(\beta_3) & -b_3\sin(\beta_2)\cos(\alpha_1) \\ 0 & 0 & 1/a_3 \end{pmatrix} . \begin{pmatrix} h \\ k \\ l \end{pmatrix}$$

Where a_i, α_i and b_i, β_i are the lattice parameters in real and reciprocal space, respectively.

Orientation Matrix

The Orientation matrix, U is used to define the sample coordinate system and transforms the B matrix into the diffractometer coordinate system.

$$U = \begin{pmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{pmatrix}$$

The full orientation can then be determined by combining all the matrices:

$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = MU(\mu).ETA(\eta).CHI(\chi).PHI(\phi).U.B. \begin{pmatrix} h \\ k \\ l \end{pmatrix}$$

Beam & Detector Orientation

In the diffractometer frame, the beam is directed along the y axis:

$$k = 2\pi/\lambda$$
$$\vec{k}^{i} = \begin{pmatrix} 0 \\ k \\ 0 \end{pmatrix}$$

Delta rotates anti-clockwise about the z-axis and gamma rotates clockwise abut the x-axis, with orientation matrices:

$$\vec{\Delta} = \begin{pmatrix} \cos(\delta) & \sin(\delta) & 0 \\ -\sin(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$\vec{\Gamma} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma)) \\ 0 & \sin(\gamma) & \cos(\gamma)) \end{pmatrix}$$

The scattered wavevector is therefore:

$$\vec{k}^{f} = k\vec{\Delta}\vec{\Gamma}\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = k\begin{pmatrix} \sin(\delta) \\ \cos(\gamma)\cos(\delta) \\ \sin(\gamma)\cos(\delta) \end{pmatrix}$$

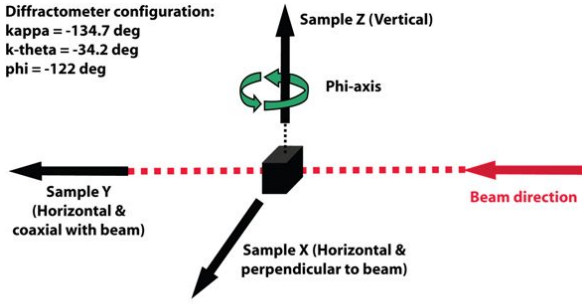
We can then index any detector angle in (h,k,l) using the UB matrix:

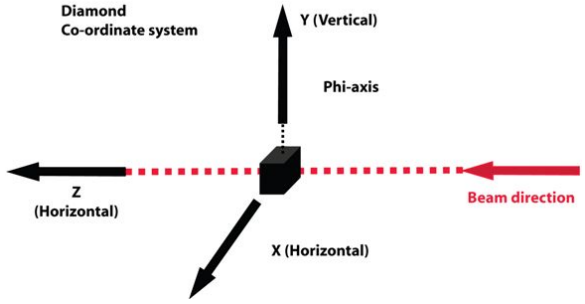
$$\begin{pmatrix} h \\ k \\ l \end{pmatrix} = (\vec{U}\vec{B})^{-1}(\vec{M}\vec{E}\vec{C}\vec{P})^{-1}(\vec{k}^{f} - \vec{k}^{i})$$

Lab Coordinate System

To transform the vector into the lab coordinate system, an additional transformation must be made:

$$\begin{pmatrix} lab_x \\ lab_y \\ lab_z \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} diffractometer_x \\ diffractometer_y \\ diffractometer_z \end{pmatrix}$$





Symbolic form

Made in Maxima

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\mu) & -\sin(\mu) \\ 0 & \sin(\mu) & \cos(\mu) \end{pmatrix}\begin{pmatrix} \cos(\eta) & \sin(\eta) & 0 \\ -\sin(\eta) & \cos(\eta) & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\chi) & 0 & \sin(\chi) \\ 0 & 1 & 0 \\ -\sin(\chi) & 0 & \cos(\chi) \end{pmatrix}\begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} \cos(\eta)\cos(\varphi)\cos(\chi) - \sin(\eta)\sin(\varphi) & \cos(\eta)\sin(\varphi)\cos(\chi) + \sin(\eta)\cos(\varphi) & \cos(\eta)\sin(\chi) \\ \sin(\mu)\cos(\varphi)\sin(\chi) + \cos(\mu)(-\sin(\eta)\cos(\varphi)\cos(\chi) - \cos(\eta)\sin(\varphi)) & \sin(\mu)\sin(\varphi)\sin(\chi) + \cos(\mu)(\cos(\eta)\cos(\varphi) - \sin(\eta)\sin(\varphi)\cos(\chi)) & -\sin(\eta)\cos(\mu)\sin(\chi) - \sin(\mu)\cos \\ \sin(\mu)(-\sin(\eta)\cos(\varphi)\cos(\chi) - \cos(\eta)\sin(\varphi)) - \cos(\mu)\cos(\varphi)\sin(\chi) & \sin(\mu)(\cos(\eta)\cos(\varphi) - \sin(\eta)\sin(\varphi)\cos(\chi)) - \cos(\mu)\sin(\varphi)\sin(\chi) & \cos(\mu)\cos(\chi) - \sin(\eta)\sin(\mu)\sin(\end{pmatrix}$$

Python Module

Diffractometer rotations

```
"""
Python module: Diffractometer Rotations

Based on definitions set by:
  Busing & Levy Acta Cryst. 22, 457 (1967)
  H. You, J. Appl. Cryst. 32 (1999), 614-623

By Dan Porter
Beamline I16, Diamond Light Source Ltd
August 2023
"""
import numpy as np


def bmatrix(a, b=None, c=None, alpha=90., beta=90., gamma=90.):
    """
    Calculate the B matrix as defined in Busing&Levy Acta Cyst. 22, 457 (1967)
    Creates a matrix to transform (hkl) into a cartesian basis:
        (qx,qy,qz)' = B.(h,k,l)'       (where ' indicates a column vector)

    The B matrix is related to the reciprocal basis vectors:
        (astar, bstar, cstar) = 2 * np.pi * B.T
    Where cstar is defined along the z axis

    :param a: lattice parameter a in Anstroms
    :param b: lattice parameter b in Anstroms
    :param c: lattice parameter c in Anstroms
    :param alpha: lattice angle alpha in degrees
    :param beta: lattice angle beta in degrees
    :param gamma: lattice angle gamma in degrees
    :returns: [3x3] array B matrix in inverse-Angstroms (no 2pi)
    """
    if b is None:
        b = a
    if c is None:
        c = a
    alpha1 = np.deg2rad(alpha)
    alpha2 = np.deg2rad(beta)
    alpha3 = np.deg2rad(gamma)

    beta1 = np.arccos( (np.cos(alpha2)*np.cos(alpha3)-np.cos(alpha1))/(np.sin(alpha2)*np.sin(alpha3)))
    beta2 = np.arccos( (np.cos(alpha1)*np.cos(alpha3)-np.cos(alpha2))/(np.sin(alpha1)*np.sin(alpha3)))
    beta3 = np.arccos( (np.cos(alpha1)*np.cos(alpha2)-np.cos(alpha3))/(np.sin(alpha1)*np.sin(alpha2)))

    b1 = 1 / (a * np.sin(alpha2) * np.sin(beta3))
    b2 = 1 / (b * np.sin(alpha3) * np.sin(beta1))
    b3 = 1 / (c * np.sin(alpha1) * np.sin(beta2))

    c1 = b1 * b2 * np.cos(beta3)
    c2 = b1 * b3 * np.cos(beta2)
    c3 = b2 * b3 * np.cos(beta1)

    bm = np.array([
        [b1, b2 * np.cos(beta3), b3 * np.cos(beta2)],
        [0, b2 * np.sin(beta3), -b3 * np.sin(beta2)*np.cos(alpha1)],
        [0, 0, 1/c]
    ])
    return bm


def rotmatrix_z(phi):
    """
    Generate diffractometer rotation matrix about z-axis (right handed)
    Equivalent to YAW in the Tain-Bryan convention
    Equivalent to -phi, -eta, -delta in You et al. diffractometer convention (left handed)
    :param phi: float angle in degrees
    :return: [3*3] array
    """
    phi = np.deg2rad(phi)
    c = np.cos(phi)
    s = np.sin(phi)
    return np.array([[c, -s, 0], [s, c, 0], [0, 0, 1]])


def rotmatrix_y(chi):
    """
```

```
    Generate diffractometer rotation matrix chi about y-axis (right handed)
    Equivalent to PITCH in the Tain-Bryan convention
    Equivalent to chi in You et al. diffractometer convention (right handed)
    :param chi: float angle in degrees
    :return: [3*3] array
    """
    chi = np.deg2rad(chi)
    c = np.cos(chi)
    s = np.sin(chi)
    return np.array([[c, 0, s], [0, 1, 0], [-s, 0, c]])


def rotmatrix_x(mu):
    """
    Generate diffractometer rotation matrix mu about x-axis (right handed)
    Equivalent to ROLL in the Tain-Bryan convention
    Equivalent to mu in You et al. diffractometer convention (right handed)
    :param mu: float angle in degrees
    :return: [3*3] array
    """
    mu = np.deg2rad(mu)
    c = np.cos(mu)
    s = np.sin(mu)
    return np.array([[1, 0, 0], [0, c, -s], [0, s, c]])


def rotmatrix_intrinsic(alpha, beta, gamma):
    """
    a rotation whose yaw, pitch, and roll angles are ,  and , respectively.
    More formally, it is an intrinsic rotation whose Tait–Bryan angles are , , , about axes z, y, x,
respectively.
    https://en.wikipedia.org/wiki/Rotation_matrix#General_rotations
    :param alpha: float yaw angle in degrees
    :param beta: float pitch angle in degrees
    :param gamma: float gamma angle in degrees
    :return: [3*3] array
    """
    alpha = np.deg2rad(alpha)
    beta = np.deg2rad(beta)
    gamma = np.deg2rad(gamma)
    ca = np.cos(alpha)
    sa = np.sin(alpha)
    cb = np.cos(beta)
    sb = np.sin(beta)
    cg = np.cos(gamma)
    sg = np.sin(gamma)
    return np.array([[ca*cb, ca*sb*sg - sa*cg, ca*sb*cg + sa*sg], [sa*cb, sa*sb*sg + ca*cg, sa*sb*cg-
ca*sg], [-sb, cb*sg, cb*cg]])


def rotmatrix_diffractometer(phi, chi, eta, mu):
    """
    an intrinsic rotation using You et al. 4S+2D diffractometer
    Z = MU.ETA.CHI.PHI
    :param phi: float left-handed rotation about z''' angle in degrees
    :param chi: float right-handed rotation about y'' angle in degrees
    :param eta: float left-handed rotation about z' angle in degrees
    :param mu: float right-handed rotation about x angle in degrees
    :return: [3*3] array
    """
    phi = np.deg2rad(phi)
    chi = np.deg2rad(chi)
    eta = np.deg2rad(eta)
    mu = np.deg2rad(mu)
    cp = np.cos(phi)
    sp = np.sin(phi)
    cc = np.cos(chi)
    sc = np.sin(chi)
    ce = np.cos(eta)
    se = np.sin(eta)
    cm = np.cos(mu)
    sm = np.sin(mu)
    return np.array([[ce*cp*cc-se*sp, ce*sp*cc+se*cp, ce*sc], [sm*cp*sc+cm*(-se*cp*cc-ce*sp),
sm*sp*sc+cm*(ce*cp-se*sp*cc), -se*cm*sc-sm*cc], [sm*(-se*cp*cc-ce*sp)-cm*cp*sc, sm*(ce*cp-se*sp*cc)-
cm*sp*sc, cm*cc-se*sm*sc]])
```

```
def diffractometer(vec, phi, chi, eta, mu):
    """
    Perform an intrinsic rotation using You et al. 4S+2D diffractometer
        mu right-handed rotation about x
        eta left-handed rotation about z'
        chi right-handed rotation about y''
        phi left-handed rotation about z'''
    Z = MU.ETA.CHI.PHI
    Angles in degrees
    vec must be 1D or column vector (3*n)

    :param vec: [3*n] vector in the sample frame
    :param phi: float angle in degrees, left handed roation about z'''
    :param chi: float angle in degrees, right handed rotation about y''
    :param eta: float angle in degrees, left handed rotation about z'
    :param mu: float angle in degrees, right handed rotation about x
    :return:  [3*n] vector in the diffractometer frame
    """
    r = np.dot(rotmatrix_x(mu), np.dot(rotmatrix_z(-eta), np.dot(rotmatrix_y(chi), rotmatrix_z(-phi))))
    return np.dot(r, vec)


def detector_wavevector(delta, gamma, wavelength):
    """
    Calculate wavevector in diffractometer axis using detector angles
    :param delta: float angle in degrees in vertical direction (about diff-z)
    :param gamma: float angle in degrees in horizontal direction (about diff-x)
    :param wavelength: float wavelength in A
    :return: [1*3] wavevector position in A-1 == kf - ki
    """

    k = 2 * np.pi / wavelength
    delta = np.deg2rad(delta)
    gamma = np.deg2rad(gamma)
    sd = np.sin(delta)
    cd = np.cos(delta)
    sg = np.sin(gamma)
    cg = np.cos(gamma)
    return k * np.array([sd, cd * cg - 1, cd * sg])


def diffractometer2hkl(ub, phi=0, chi=0, eta=0, mu=0, delta=0, gamma=0, wavelength=1.0):
    """
    Return [h,k,l] position of diffractometer axes with given UB and wavelength
    :param ub: [3*3] array UB orientation matrix following Busing & Levy
    :param phi: float sample angle in degrees
    :param chi: float sample angle in degrees
    :param eta: float sample angle in degrees
    :param mu: float sample angle in degrees
    :param delta: float detector angle in degrees
    :param gamma: float detector angle in degrees
    :param wavelength: float wavelength in A
    :return: [h,k,l]
    """
    q = detector_wavevector(delta, gamma, wavelength)  # You Ql (12)
    z = np.dot(rotmatrix_x(mu), np.dot(rotmatrix_z(-eta), np.dot(rotmatrix_y(chi), rotmatrix_z(-phi))))
# You Z (13)

    inv_ub = np.linalg.inv(ub)
    inv_z = np.linalg.inv(z)

    hphi = np.dot(inv_z, q)
    return np.dot(inv_ub, hphi).T


if __name__ == '__main__':
    # Test functions
    b = 2 * np.pi * bmatrix(a=2.85, c=10.8, gamma=120.)
    u = np.eye(3)
    ub = np.dot(u, b)

    wavelength = 12.3984 / 8  # energy in keV, wavelength in Angstroms

    # calcualte expected two-theta
```

```
d_space = 10.8 / 6  # (0,0,6)
tth = 2 * np.rad2deg(np.arcsin(wavelength / (2 * d_space)))  # deg

# (006) vertical geometry
v_hkl = diffractometer2hkl(
    ub=ub,
    phi=0,
    chi=90,
    eta=tth / 2,
    mu=0,
    delta=tth,
    gamma=0,
    wavelength=wavelength
)
# Horizontal geometry
h_hkl = diffractometer2hkl(
    ub=ub,
    phi=0,
    chi=0,
    eta=0,
    mu=tth/2,
    delta=0,
    gamma=tth,
    wavelength=wavelength
)

np.set_printoptions(precision=4, suppress=True, linewidth=200)
print('Wavelength: %.3f A' % wavelength)
print('Two-Theta: %.3f Deg' % tth)
print('  Vertical geometry hkl: %s' % v_hkl)
print('Horizontal geometry hkl: %s' % h_hkl)
```