Manual to fit CTR/RAXS data for muscovite (001)-oligomer system

1. Purpose of this template script

To easily set up structural model for CTR/RAXS data fitting based on muscovite (001) substrate. The main control script was written in a concise and user-friendly way, so that the control flow could be easily followed. The script also has a lot of flexibility to expand its current available functionality to consider more complicated cases. The user will only need to write some function as a plug-in to be called inside the main script.

1. Feature of this template script

Specifically relevant to muscovite(001) substrate. It was initially developed to model Zr oligomer adsorption on the substrate surface, although it can be used to model other aqueous species in both monomer and oligomer forms.

The isomorphic substitution of Al to Si sites is considered by including two domains as a complete system, ie 75% of Si-only surface + 25% of Al-only surface. You could define multiple domain couples to set up a more detailed system, but make sure to enforce the weight percentage constraints for those constituent domains within each domain couple to make the structure built physically reasonable.

The bond valence constraint is not yet available in the script. The feature will be added when it is necessary.

The addition of sorbates (including adsorbed water) is based on the symmetry operation observed at the substrate surface, ie two symmetry sites related by diagonal glide plane symmetry with coordinates of (x,y,z) and (x+0.5, y+0.5, z), respectively.

You can consider multiply sets of sorbates (to be fit independently), but there is only one set of diffused layered structure (water and sorbate) throughout the whole interfacial area.

1. How to setup a model? A tour of the main script.

**##<coordination system definition>##**

x0\_v,y0\_v,z0\_v=np.array([1.,0.,0.]),np.array([0.,1.,0.]),np.array([0.,0.,1.])

#anonymous function f1 calculating transforming matrix with the basis vector expressions,x1y1z1 is the original basis vector

#x2y2z2 are basis of new coor defined in the original frame,new=T.orig

f1=lambda x1,y1,z1,x2,y2,z2:np.array([[np.dot(x2,x1),np.dot(x2,y1),np.dot(x2,z1)],\

[np.dot(y2,x1),np.dot(y2,y1),np.dot(y2,z1)],\

[np.dot(z2,x1),np.dot(z2,y1),np.dot(z2,z1)]])

BASIS=np.array([5.1988, 9.0266, 20.1058])

BASIS\_SET=[[1,0,0],[0,1,0],[0.10126,0,1.0051136]]

T=inv(np.transpose(f1(x0\_v,y0\_v,z0\_v,\*BASIS\_SET)))

T\_INV=inv(T)

You don’t need any edit here. We define the transformation matrix to be used to convert a coordinate in the muscovite lattice system to a Cartesian system (orthogonal), so that the geometric calculation can be relatively easily undertaken. The following two equations are valid based on these definitions.

Coordinates\_in\_Cartesian\_system=np.dot(T,coordinates\_in\_lattice\_system)

Coordinates\_in\_lattice\_system=np.dot(T,coordinates\_in\_Cartesian\_system)

**##<global handles>##**

RUN=False##to be set##

BATCH\_PATH\_HEAD='P:\\apps\\genx\_pc\_qiu\\batchfile\\'##to be set##

OUTPUT\_FILE\_PATH='D:\\'

F1F2=np.loadtxt(BATCH\_PATH\_HEAD+'f1f2\_temp.f1f2')

E0=13000##to be set##(absorbtion edge in eV)

NUMBER\_RAXS\_SPECTRA=28##to be set##

NUMBER\_DOMAIN=2##to be set##

COHERENCE=True

RAXR\_EL='Zr'##to be set##

RAXR\_FIT\_MODE='MD'##to be set##

HEIGHT\_OFFSET=-2.6685#if set to 0, the top atomic layer is at 2.6685 in fractional unit before relaxation

INFO\_LIB={'basis':BASIS,'sorbate\_el':'Zr','coordinate\_el':'O','T':T,'T\_INV':T\_INV,'oligomer\_type':'tetramer'}##to be set##

Here we define very important global variables for model setup and model fitting.

RUN: if true then all IO functions will not be executed.

BATCH\_PATH\_HEAD: the location of the batchfile folder, which contains bulk and surface files.

OUTPUT\_FILE\_PATH: You specify the location to save output files.

F1F2:The file of resonant anomalous correction items (two columns)

E0: The absorbtion energy of the element of interest

NUMBER\_RAXS\_SPECTRA: How many RAXR spectra in the full dataset.

NUMBER\_DOMAIN: How many domains to be considered (should be even number)?

COHERENCE: How do you want to add up the structure factor from each domain?

RAXR\_EL: Element symbol of resonant element

RAXR\_FIT\_MODE: either ‘MD’ for model-dependent fit or ‘MI’ for model-independent fit.

HEIGHT\_OFFSET: an extra offset (fractional basis) to the z values of all atoms. It will not have any effect on the structure factor, but it will change the profile of Fourier components A and P as a function of Q. It is better to set the terminal surface atom to have 0 z values.

INFO\_LIB: a library to define the model features. You may define multiple such libraries (use different names, INFO\_LIB1, INFO\_LIB2…), if you have domain couples not sharing the same setup. Eg, one domain couple has a ‘tetramer’ type, while another one has a ‘hexamer’ oligomer type. The ‘oligomer\_type’ can be one of ‘monoer’, ‘tetramer’, ‘hexamer’ and ‘decamer’.

**##<setting slabs>##**

unitcell = model.UnitCell(5.1988, 9.0266, 20.1058, 90, 95.782, 90)

inst = model.Instrument(wavel = .833, alpha = 2.0)

bulk = model.Slab(T\_factor='u')#bulk

domain\_creator.add\_atom\_in\_slab(bulk,BATCH\_PATH\_HEAD+'muscovite\_001\_bulk.str',height\_offset=HEIGHT\_OFFSET)

Domain1 = model.Slab(c = 1.0,T\_factor='u')#surface slabs-Domain1

domain\_creator.add\_atom\_in\_slab(Domain1,BATCH\_PATH\_HEAD+'muscovite\_001\_surface\_Al.str',attach='\_D1',height\_offset=HEIGHT\_OFFSET)

Domain2 = model.Slab(c = 1.0,T\_factor='u')#surface slabs-Domain2

domain\_creator.add\_atom\_in\_slab(Domain2,BATCH\_PATH\_HEAD+'muscovite\_001\_surface\_Si.str',attach='\_D2',height\_offset=HEIGHT\_OFFSET)

Here we define the bulk and surface slabs (Domain1 and Domain2). You can copy and past to define more domain couples. You will need to change the attach argument accordingly if you have more than one domain couple.

**##<Adding sorbates>##**

#domain1

rgh\_domain1=UserVars()

geo\_lib\_domain1={'cent\_point\_offset\_x':0,'cent\_point\_offset\_y':0,'cent\_point\_offset\_z':0,'r':2.2,'theta':59.2641329,'rot\_x':0,'rot\_y':0,'rot\_z':0}

Domain1,rgh\_domain1=domain\_creator.add\_sorbate(domain=Domain1,anchored\_atoms=[],func=domain\_creator\_sorbate.OS\_sqr\_antiprism\_oligomer,geo\_lib=geo\_lib\_domain1,info\_lib=INFO\_LIB,domain\_tag='\_D1',rgh=rgh\_domain1,index\_offset=[0,1],height\_offset=HEIGHT\_OFFSET)

#domain2

rgh\_domain2=UserVars()

geo\_lib\_domain2={'cent\_point\_offset\_x':0,'cent\_point\_offset\_y':0,'cent\_point\_offset\_z':0,'r':2.2,'theta':59.2641329,'rot\_x':0,'rot\_y':0,'rot\_z':0}

Domain2,rgh\_domain2=domain\_creator.add\_sorbate(domain=Domain2,anchored\_atoms=[],func=domain\_creator\_sorbate.OS\_sqr\_antiprism\_oligomer,geo\_lib=geo\_lib\_domain2,info\_lib=INFO\_LIB,domain\_tag='\_D2',rgh=rgh\_domain2,index\_offset=[0,1],height\_offset=HEIGHT\_OFFSET)

Here we add sorbates (and the associated coordinative members) to the surface slabs. At the same time, we also define the user-defined variable instance for each domain. The content of geo\_lib\_domain really depend on the function you want to use for adding sorbates. You could add more than one set of sorbate to each domain, then you need to change the index\_offset ([0,1] for the first set and [2,3] for the second set and so on). If you have different INFO\_LIB for different domains, then change it accordingly. If you want to add sorbate in a different way, you will only need to write a function inside the domain\_creator\_sorbate module and pass it to the func argument of add\_sorbate function.

**##<Adding absorbed water>##to be set##**

#domain1

Domain1,absorbed\_water\_pair1\_D1=domain\_creator.add\_oxygen\_pair\_muscovite(domain=Domain1,ids=['O1a\_W\_D1','O1b\_W\_D1'],coors=np.array([[0,0,2.2+HEIGHT\_OFFSET],[0.5,0.5,2.2+HEIGHT\_OFFSET]]))

Domain1,absorbed\_water\_pair2\_D1=domain\_creator.add\_oxygen\_pair\_muscovite(domain=Domain1,ids=['O2a\_W\_D1','O2b\_W\_D1'],coors=np.array([[0,0,2.3+HEIGHT\_OFFSET],[0.5,0.5,2.3+HEIGHT\_OFFSET]]))

#domain2

Domain2,absorbed\_water\_pair1\_D2=domain\_creator.add\_oxygen\_pair\_muscovite(domain=Domain2,ids=['O1a\_W\_D2','O1b\_W\_D2'],coors=np.array([[0,0,2.2+HEIGHT\_OFFSET],[0.5,0.5,2.2+HEIGHT\_OFFSET]]))

Domain2,absorbed\_water\_pair2\_D2=domain\_creator.add\_oxygen\_pair\_muscovite(domain=Domain2,ids=['O2a\_W\_D2','O2b\_W\_D2'],coors=np.array([[0,0,2.3+HEIGHT\_OFFSET],[0.5,0.5,2.3+HEIGHT\_OFFSET]]))

Here we define the adsorbed waters, you will need to specify the associated ids and the corresponding coordinates. The function will update the domain instance and return a group for each set of adsorbed water.

**##<Define atom groups>##**

#surface atoms

group\_number=5##to be set##(number of groups to be considered for model fit)

.

.

.

atom\_group\_info=[{'domain':Domain1,'ref\_id\_list':ref\_id\_list\_Al[0:group\_number],'ref\_group\_names':ref\_group\_names\_Al[0:group\_number],'ref\_sym\_list':ref\_sym\_list[0:group\_number],'domain\_tag':'\_D1'},

{'domain':Domain2,'ref\_id\_list':ref\_id\_list\_Si[0:group\_number],'ref\_group\_names':ref\_group\_names\_Si[0:group\_number],'ref\_sym\_list':ref\_sym\_list[0:group\_number],'domain\_tag':'\_D2'}]

Here we define the atom groups for surface atoms (atoms related by symmetry will grouped together). You will need to specify the group\_number (number of total atom groups counting from terminal oxygen groups) and also expand the atom\_group\_info if you have more than one domain couple (change the value of ‘domain’ key and the value of ‘domain\_tag’ according to your setup).

#sorbate\_atoms

sorbate\_id\_list\_domain1=[[id for id in Domain1.id if INFO\_LIB['sorbate\_el'] in id],[id for id in Domain1.id if INFO\_LIB['sorbate\_el'] in id and 'O' not in id],[id for id in Domain1.id if INFO\_LIB['sorbate\_el'] in id and 'O' in id]]

sorbate\_id\_list\_domain2=[[id for id in Domain2.id if INFO\_LIB['sorbate\_el'] in id],[id for id in Domain2.id if INFO\_LIB['sorbate\_el'] in id and 'O' not in id],[id for id in Domain2.id if INFO\_LIB['sorbate\_el'] in id and 'O' in id]]

sorbate\_sym\_list\_domain1=[]

sorbate\_sym\_list\_domain2=[]

sorbate\_group\_names\_domain1=['sorbate\_D1',INFO\_LIB['sorbate\_el']+'\_D1','HO\_D1']

sorbate\_group\_names\_domain2=['sorbate\_D2',INFO\_LIB['sorbate\_el']+'\_D2','HO\_D2']

sorbate\_atom\_group\_info=[{'domain':Domain1,'ref\_id\_list':sorbate\_id\_list\_domain1,'ref\_group\_names':sorbate\_group\_names\_domain1,'ref\_sym\_list':sorbate\_sym\_list\_domain1,'domain\_tag':''},

{'domain':Domain2,'ref\_id\_list':sorbate\_id\_list\_domain2,'ref\_group\_names':sorbate\_group\_names\_domain2,'ref\_sym\_list':sorbate\_sym\_list\_domain2,'domain\_tag':''}]

sorbate\_groups,sorbate\_group\_names=domain\_creator.setup\_atom\_group(gp\_info=sorbate\_atom\_group\_info)

Here we define the sorbate groups. Expand this accordingly for a multi-domain system (mostly copy and paste work). There are three sorbate groups defined within each domain, ie sorbate\_D1, Zr\_D1, HO\_D1 (domain1 and Zr sorbate as an example). Sorbate\_D1 include all sorbates and coordinative members (used to set occupancy), Zr\_D1 include all sorbates (used to define thermal factor u) and HO\_D1 include all coordinative members (used to define thermal factor u independent on those for sorbate).

**##<Define other pars>##**

rgh=domain\_creator.define\_global\_vars(rgh=UserVars(),domain\_number=NUMBER\_DOMAIN)#global vars

rgh\_raxs=domain\_creator.define\_raxs\_vars(rgh=UserVars(),number\_spectra=NUMBER\_RAXS\_SPECTRA,number\_domain=NUMBER\_DOMAIN)#RAXR spectra pars

rgh\_dlw=domain\_creator.define\_diffused\_layer\_water\_vars(rgh=UserVars())#Diffused Layered water pars

rgh\_dls=domain\_creator.define\_diffused\_layer\_sorbate\_vars(rgh=UserVars())#Diffused Layered sorbate pars

Here we define other parameters needed for the structure factor calculation. Rgh has attributes of domain weight and roughness. Rgh\_raxs contains all the variables needed to do model-independent fit (a, b, c, A, P). And note each domain has its own RAXS pars. For example, a1\_D1 is the ‘a’ for the first spectra corresponding to domain1. Rgh\_dlw define the pars for diffused layered water and rgh\_dls define the pars for diffused layered sorbate.

**##<make fit table file>##**

if not RUN:

table\_container=[]

rgh\_instance\_list=[rgh]+groups+[absorbed\_water\_pair1\_D1,absorbed\_water\_pair2\_D1,absorbed\_water\_pair1\_D2,absorbed\_water\_pair2\_D2]+\

sorbate\_groups+[rgh\_domain1,rgh\_domain2,rgh\_dlw,rgh\_dls]

rgh\_instance\_name\_list=['rgh']+group\_names+['absorbed\_water\_pair1\_D1','absorbed\_water\_pair2\_D1','absorbed\_water\_pair1\_D2','absorbed\_water\_pair2\_D2']+\

sorbate\_group\_names+['rgh\_domain1','rgh\_domain2','rgh\_dlw','rgh\_dls']

table\_container=make\_grid.set\_table\_input\_all(container=table\_container,rgh\_instance\_list=rgh\_instance\_list,rgh\_instance\_name\_list=rgh\_instance\_name\_list,par\_file=BATCH\_PATH\_HEAD+'pars\_ranges.txt')

#raxs pars

table\_container=make\_grid.set\_table\_input\_raxs(container=table\_container,rgh\_group\_instance=rgh\_raxs,rgh\_group\_instance\_name='rgh\_raxs',par\_range={'a':[0,1],'b':[0,1],'c':[0,1],'A':[0,1],'P':[0,1]},number\_spectra=NUMBER\_RAXS\_SPECTRA,number\_domain=NUMBER\_DOMAIN)

#build up the tab file

make\_grid.make\_table(container=table\_container,file\_path=OUTPUT\_FILE\_PATH+'par\_table.tab')

Here we generate parameter table to be imported. Expand this accordingly when considering more than one domain couple.

**def Sim(data):**

##<Extract pars>##

layered\_water\_pars=vars(rgh\_dlw)

layered\_sorbate\_pars=vars(rgh\_dls)

raxs\_vars=vars(rgh\_raxs)

##<update sorbates>##

domain\_creator.update\_sorbate(domain=Domain1,anchored\_atoms=[],func=domain\_creator\_sorbate.OS\_sqr\_antiprism\_oligomer,info\_lib=INFO\_LIB,domain\_tag='\_D1',rgh=rgh\_domain1,index\_offset=[0,1],height\_offset=HEIGHT\_OFFSET)#domain1

domain\_creator.update\_sorbate(domain=Domain2,anchored\_atoms=[],func=domain\_creator\_sorbate.OS\_sqr\_antiprism\_oligomer,info\_lib=INFO\_LIB,domain\_tag='\_D2',rgh=rgh\_domain2,index\_offset=[0,1],height\_offset=HEIGHT\_OFFSET)#domain2

#You can add more domains

##<format domains>##

domain={'domains':[Domain1,Domain2],'layered\_water\_pars':layered\_water\_pars,'layered\_sorbate\_pars':layered\_sorbate\_pars,\

'global\_vars':rgh,'raxs\_vars':raxs\_vars,'F1F2':F1F2,'E0':E0,'el':RAXR\_EL}

sample = model.Sample(inst, bulk, domain, unitcell,coherence=COHERENCE,surface\_parms={'delta1':0.,'delta2':0.})

Inside sim function, you only need to update\_sorbate and define domain accordingly based on your model structure.

1. How to export files for structure visualization?

Set RUN to False, all the files will be exported. For AP over Q profiles, you will need one more step to be done inside the shell. The command is model.script\_module.create\_plots.append\_errors\_for\_A\_P(par\_instance=model.parameters,dump\_file='D://temp\_plot\_raxr\_A\_P\_Q',raxs\_rgh='rgh\_raxs'). It will extract the model errors for AP and append them to the saved file (temp\_plot\_raxr\_A\_P\_Q).

1. How to plot model files?

Use Python shell, execute the script file (exefile(‘P:\apps\genx\_pc\_qiu\supportive\_functions\create\_plots.py’)). You may need to edit the script to select for plots to be made. Just edit the first code line after *if \_\_name\_\_=="\_\_main\_\_"* in the script: