Tutorial: Rosetta tools for structure determination in density

Ray Y.-R. Wang, Frank DiMaio
Keystone Conference on Hybrid Methods
March 2015

This tutorial is intended to introduce users to several different ways Rosetta may be used to solve various structure determination tasks given low-resolution cryoEM density and X-ray crystallographic data. It is not intended to replace the user's guide, available at https://www.rosettacommons.org/manuals/latest/main/.

The tutorial is split up into four parts: an introduction to Rosetta and the density tools in Rosetta, and three main scenarios in which Rosetta may be used to aid in structure determination against low-resolution experimental density data. Scenario 1 describes model refinement against low-resolution EM density, Scenario 2 describes model rebuilding, where one wishes to rebuild missing regions of a structure (for example, from a homology model), and Scenario 3 describes *de novo* model building, where we only have density data, and a sequence, with no identifiable structural homologues. A final section describes how these protocols may also be used for model building and refinement against unphased crystallographic data.

In each scenario, we present the most basic usage of Rosetta for the task, and then describe additional options that may be useful. Command-line flags and input scripts are provided in shaded boxes, with boldfaced text indicating parameters of note. These parameters are described in the text following the command line.

Additionally, this tutorial focuses on refinement of structures when density data is available, however, several of the tools presented (in particular, those presented in Scenario 2) are useful tools even when no density data is available. In these cases, turning off density scoring terms will enable structure prediction in the absence of experimental data.

Finally, most of the applications described in this tutorial work as given in the current Rosetta release, version 3.5. However, there are several options that make use of applications unavailable in the current version. These will be available in the next weekly release of Rosetta. These options are pointed out in the documentation; they will be available in the next release of Rosetta, though the option names are not guaranteed to stay the same.

Scenario 0: Rosetta and electron density basics

This section provides a brief introduction to using Rosetta, and an overview of using density data within Rosetta.

Density scoring terms in Rosetta

Agreement to density is implemented in Rosetta as an additional energy term. Rosetta assesses agreement to density by computing the density that one would expect to see, given a model, and measuring the agreement of the expected and experimental density.

In general, there are two different fit-to-density implementations that are relevant, a slow-but-accurate version, and a fast-but-less-accurate version:

elec_dens_window
: Recommended only for medium-high resolution (<4Å), and only in the final stages of refinement. Uses a "windowed correlation" over overlapping windows of residues. The score is derived from this correlation (based on the probability of seeing a particular correlation in a correct conformation).

elec_dens_fast
: Recommended for use in all other cases. Uses interpolation on a precomputed grid of per-atom scores to approximate the high-resolution scoring function. This version is significantly faster (~10x) and is very highly correlated the expensive version above.

These energy terms may be provided to Rosetta in three ways. First, it may be placed in a *patch file* like any other scoring term in Rosetta:

```
elec_dens_fast=2.0
elec_dens_window=20.0
```

This is passed to Rosetta with the flag `-score:patch` *`patchfile`*. Secondly, it may be provided in a RosettaScript XML file as input (see the RosettaScript documentation or Scenarios 1 & 2 for examples of this):

```
<Reweight scoretype=elec_dens_fast weight=2.0/>
<Reweight scoretype=elec_dens_window weight=20.0/>
```

Finally, the following flags may control the two scoring functions, respectively:

```
-edensity:sliding_window_wt 2.0
-edensity:fast_dens_wt 20.0
```

The recommended weights for each of these terms vary depending on the density map resolution, starting model quality, and protocol. Scenario 2 describes one way in which these weights may be tuned. However, the following are good rules of thumb for setting the density weight within Rosetta:

elec_dens_window – a weight of **2.0** is generally reasonable
elec_dens_fast – a weight of **20** is generally reasonable

In both cases, the weight should be reduced by ~ a factor of two for very low-resolution or noisy density, and should be increased by ~ a factor of two for very high-resolution (sub-3.5Å) density.

Additionally, if the sliding window scoring function is used, the additional flag `-density:sliding_window n` should also be provided, which gives the width (in residues) of the widow to use. This should always be an odd number; 3 is recommended.

In addition to the score terms above, there are also several flags that control map scoring behavior. Maps are read into Rosetta using either the flag:

```
-edensity::mapfile mapfile.mrc
```

Or from XML:

```
<LoadDensityMap name=loaddens mapfile=mapfile.mrc/>
```

Maps may be in either CCP4 or MRC format (the map type is automatically detected from the header info).

The resolution of the map, used when comparing calculated to experimental density, is specified with the flag:

```
-edensity::mapreso 5.0
```

Maps may also be resampled to reduce memory usage and runtime. This is done through the flag:

```
-edensity::grid_spacing 2.0
```

Notice that this flag should *never be more than half the given resolution*, and if using the fast scoring function *never more than a third of the resolution*. For both parameters, the default is generally fine (don't resample, and assume the resolution is ~3x the grid sampling).

Finally, one may choose to calculate density using either cryoEM or X-ray scattering factors. At low resolution, this probably makes little difference, but might at resolutions better than about 3.5Å. The default is to use X-ray scattering factors, to turn on cryoEM scattering factors instead, use the following flag:

```
-edensity::cryoem_scatterers
```

Example 0A: Scoring a PDB in Rosetta with density

Most simply, one may wish to simply score a model using Rosetta. This is most easily accomplished using the *score_jd2* application. A sample command line to rescore the structure in density is given in *scenario0_rosetta_basics/ex_A_run_rescore.sh*. It illustrates the use of various density flags to provide Rosetta with experimental density information.

```
$ROSETTA3/source/bin/score_jd2.macosclangrelease \
 -database $ROSETTA3/database/ \
 -database ~/Rosetta/database/ \
 -in::file::s 1isrA.pdb 1issA.pdb \
 -ignore_unrecognized_res \
 -edensity::mapfile 1issA_6A.mrc \
 -edensity::mapreso 5.0 \
 -edensity::grid_spacing 2.0 \
 -edensity::fastdens_wt 20.0 \
 -edensity::cryoem_scatterers \
 -crystal_refine
```

Some flags of note are boldfaced above. First, the input structure is provided with the command `-in::file::s`. This is common to many Rosetta applications, and more than one input may be provided;

each will be processed independently. The flags beginning with `-edensity::` tell Rosetta about the density map into which it is being fit. The name of the mapfile (in CCP4 or MRC format), the resolution of the map, the grid sampling of the map (*which should never be more than half the resolution*), and the weights on the various fit-to-density scoring functions. These same flags are reused for many different protocols in addition to relax. Finally, the flag `-crystal_refine` the flag turns on several density-related options related to PDB reading and writing, and should always be used when refining against density data.

Note: The input PDB must be aligned to the density map using some external tool. Rosetta will optionally rigid-body minimize the structure into density before rescoring by providing the flag `-edensity::realign min` to the command. If this is done, the flag `-out::pdb` will write the minimized PDB file to a PDB file.

This command line outputs a score file, *score.sc,* that gives, for each structure specified with `-in::file::s` – the score with respect to each term in Rosetta's energy function. The meaning of some of the othe terms are shown below.

fa_atr, fa_rep: Lennard-Jones attractive, repulsive energies
fa_sol: Lazaridis-Karplus solvation energy
pro_close: proline ring closure energy
hbond_*: hydrogen bond energy terms
dslf_fa13: disulfide bond energy term
rama, omega: Ramachandran preferences**,** omega angle preferences
fa_dun: internal energy of sidechain rotamers as derived from Dunbrack's statistics.
p_aa_pp: probability of observing a particular amino acid given phi/psi angles
ref: reference energy for each amino acid

Example 0B: Simple refinement into density using RosettaScripts and relax

In this section we introduce RosettaScripts by way of a very simple refinement-into-density example. RosettaScripts provides an XML scripting interface to Rosetta that allows fine-grained control of protocols. The syntax is fully described in the documentation; however, a very brief introduction is provided here. The basic syntax for the XML is illustrated here (*scenario0_rosetta_basics/ex_B1_relax_density.xml*)

Declare score functions

Declare movers (atomic conformation operations)

Protocol is a sequence of movers

```
<ROSETTASCRIPTS>
   <SCOREFXNS>
      <dens weights=talaris2013_cart>
         <Reweight scoretype=elec_dens_fast weight=20.0/>
      </dens>
   </SCOREFXNS>

   <MOVERS>
       <SetupForDensityScoring name=setupdens/>
       <LoadDensityMap name=loaddens mapfile="1issA_6A.mrc"/>
       <FastRelax name=relaxcart scorefxn=dens repeats=1 cartesian=1/>
   </MOVERS>

   <PROTOCOLS>
      <Add mover=setupdens/>
      <Add mover=loaddens/>
      <Add mover=relaxcart/>
   </PROTOCOLS>
   <OUTPUT scorefxn=dens/>
</ROSETTASCRIPTS>
```

In this particular example, we declare a single scorefunction, *dens*, which is set to use the weights *talaris2013_cart* (a default score function, don't need to worry about it), and also turns on *elec_dens_fast*, with a weight of 20. We then declare three movers, *SetupForDensityScoring*, *LoadDensityMap*, and *FastRelax*, which sets up the loaded structure for density scoring, loads a map into memory, and then refines the structure using the FastRelax protocol. The declared scorefunction, *dens*, is used as an input to the *FastRelax* mover.

To run this script, we use the following command line (*scenario0_rosetta_basics/ex_B1_relax_density.sh)*:

```
$ROSETTA3/source/bin/rosetta_scripts.macosclangrelease \
 -database $ROSETTA3/database/ \
 -in::file::s 1isrA.pdb \
 -parser::protocol ex_B1_run_RS_relax_density.xml \
 -ignore_unrecognized_res \
 -edensity::mapreso 5.0 \
 -edensity::cryoem_scatterers \
 -crystal_refine \
 -out::suffix _relax \
 -default_max_cycles 200
```

Note: We do not have to specify the density weight or the map file on the command line, since they are handled within the XML file. However, other density options must be specified on the command line. When using RosettaScripts, the density weights *must* be specified in the XML, the input map may be specified *either way*.

Finally, in the previous XML file, the tag *cartesian=1* appears, which refines the structure in Cartesian space. Rosetta also allows refinement in torsional space, which may be better for capturing domain motion, and for further reduction in model parameters against low-resolution data. To enable torsional refinement (*scenario0_rosetta_basics/ex_B1_relax_tors_density.xml*), we make two small changes to the XML:

```
…
<dens weights=talaris2013>
…
<FastRelax name=relaxtors scorefxn=dens repeats=1 cartesian=0/>
…
```

Scenario 1: Model refinement via iterative local rebuilding

In this scenario, we introduce our basic cryoEM refinement protocol, which uses an iterative local rebuilding procedure to escape local minima during refinement. We additionally introduce the following tools:

- Refinement of symmetrical systems and *make_symmdef_file*
- Atomic B factor fitting
- Assessing model-map agreement with *density_tools*

As a running example, we refine models of the transmembrane region of the TRPV1 ion channel, using a 3.4 Å cryoEM single particle reconstruction (M. Liao, E. Cao, D. Julius, Y. Cheng, *Nature*, 2013), and the deposited model (id: 3j5p) as a starting model. We will first refine this asymmetrically, and then introduce symmetric refinement.

Example 1A: Asymmetric refinement into cryoEM density

A summary of the XML used for refinement (*scenario1_cryoem_refinement/ex_A1_asymm_1cycle.xml*) is shown below. Following, a brief description of the movers and options available is provided.

```
<ROSETTASCRIPTS>
   <SCOREFXNS>
      <cen weights="score4_smooth_cart">
         <Reweight scoretype=elec_dens_fast weight=20/>
      </cen>

      <dens_soft weights="soft_rep">
         <Reweight scoretype=cart_bonded weight=0.5/>
         <Reweight scoretype=pro_close weight=0.0/>
         <Reweight scoretype=fa_sol weight=0.0/> # membrane protein
         <Reweight scoretype=elec_dens_fast weight=25/>
      </dens_soft>

      <dens weights=talaris2013_cart>
         <Reweight scoretype=elec_dens_fast weight=25/>
         <Reweight scoretype=fa_sol weight=0.0/> # membrane protein
         <Set scale_sc_dens_byres="R:0.76,K:0.76,E:0.76,D:0.76,M:0.76,
            C:0.81,Q:0.81,H:0.81,N:0.81,T:0.81,S:0.81,Y:0.88,W:0.88,
            A:0.88,F:0.88,P:0.88,I:0.88,L:0.88,V:0.88"/>
      </dens>
   </SCOREFXNS>
   <MOVERS>
       <SetupForDensityScoring name=setupdens/>
       <LoadDensityMap name=loaddens mapfile="./trpv1_half1.mrc"/>

       <MinMover name=cenmin scorefxn=cen type=lbfgs_armijo_nonmonotone
            max_iter=200 tolerance=0.00001 bb=1 chi=1 jump=ALL/>

       <CartesianSampler name=cen5_50 automode_scorecut=-0.5 scorefxn=cen
            mcscorefxn=cen fascorefxn=dens_soft strategy="auto" fragbias="density"
            rms=2.0 ncycles=200 fullatom=0 bbmove=1 nminsteps=25 temp=4 fraglens=3
            nfrags=25/>

       <FastRelax name=relaxcart scorefxn=dens repeats=1 cartesian=1/>
    </MOVERS>
```

```
    <PROTOCOLS>
        <Add mover=setupdens/>
        <Add mover=loaddens/>
        <Add mover=cenmin/>
        <Add mover=relaxcart/>
        <Add mover=cen5_50/>
        <Add mover=relaxcart/>
        <Add mover=relaxcart/>
    </PROTOCOLS>
    <OUTPUT scorefxn=dens/>
</ROSETTASCRIPTS>
```

The protocol is somewhat similar to the relax protocol of Scenario 0B, and in fact, calls the relax mover three times. However, there are a few new additions, as well as a few scorefunction changes, highlighted in bold.

The main addition is the *CartesianSampler* mover. This mover identifies backbone segments it believes are incorrect, given local strain and local density agreement. A Z score is computed compared to other refined near-atomic-resolution structures, and anything with a Z score worse than -0.5 is selected for rebuilding (this cutoff value may be changed through the *automode_scorecut* flag). The rebuilding loop then uses predicted local backbones to replace these segments iteratively. The number of rebuilding cycles can be controlled with the *ncycles* flag.

Two other flags that might be useful to modify are the *fraglens* flag and the *rms* flag. These control the length of backbone segment that gets replaced, and the RMS deviation of the fragment endpoints necessary to accept a fragment, respectively. Increasing both of these parameters will increase the local movement of the model during refinement. Finally, the remaining flags control acceptance behavior and/or sampling behavior, and for most cases should be left as-is.

An additional option is passed to the density scoring via the <*Set scale_sc_dens_byres*=.../> tag. In the refinement protocol, this sets a per-amino-acid sidechain reweighing. A value of 1 means the sidechain density weight is equal to the backbone. The weights shown in this example were determined by fitting these parameters into refined structures into several 3-5Å cryoEM density maps, the end result is a slight downweighing of sidechain density, particularly for charged sidechains.

Finally, the *MinMover* first minimizes the structure using a low-resolution energy function (*cen*). We have found this step is most useful for improving protein backbone geometry, particularly with hand-traced models. This low-resolution scorefunction uses the *centroid* representation, and requires that either the flag `-in::file::centroid_input` is provided, or the *SwitchResidueTypeSet* mover is first called (in this example we use the former option).

Note: Older versions of Rosetta do not support some of these options. For these versions, remove the <*Set scale_sc_dens_byres*=.../> and replace the <*CartesianSampler* .../> tag with (*scenario1_cryoem_refinement/ex_A1_asymm_1cycle_legacy.xml*):

```
        <CartesianSampler name=cen5_50 scorefxn=cen mcscorefxn=cen
             fascorefxn=dens_soft strategy="density" fragbias="density" rms=2.0
             ncycles=200 fullatom=0 bbmove=1 nminsteps=25 temp=4 fraglens=3
             nfrags=25/>
```

While this script shows one cycle of rebuilding, in practice, we have found that multiple cycles, with increasing strictness on the Z score cutoff (*automode_scorecut*) works much better in practice, as we fix the regions with worst local strain and local density fit, and then fit the more borderline cases. Thus, in practice, we will use an XML like the following (*scenario1_cryoem_refinement/ ex_A2_asymm_multicycle.xml*):

```
<CartesianSampler name=cen5_50 automode_scorecut=-0.5 scorefxn=cen
    mcscorefxn=cen fascorefxn=dens_soft strategy="auto" fragbias=density rms=2.0
    ncycles=200 fullatom=0 bbmove=1 nminsteps=25 temp=4 />
<CartesianSampler name=cen5_60 automode_scorecut=-0.3 scorefxn=cen
    mcscorefxn=cen fascorefxn=dens_soft strategy="auto" fragbias=density rms=1.5
    ncycles=200 fullatom=0 bbmove=1 nminsteps=25 temp=4 />
<CartesianSampler name=cen5_70 automode_scorecut=-0.1 scorefxn=cen
    mcscorefxn=cen fascorefxn=dens_soft strategy="auto" fragbias=density rms=1.5
    ncycles=200 fullatom=0 bbmove=1 nminsteps=25 temp=4 />
<CartesianSampler name=cen5_80 automode_scorecut=0.0 scorefxn=cen
    mcscorefxn=cen fascorefxn=dens_soft strategy="auto" fragbias=density rms=1.0
    ncycles=200 fullatom=0 bbmove=1 nminsteps=25 temp=4 />
```

With a protocols section:

```
…
<Add mover=cen5_50/>
<Add mover=relaxcart/>
<Add mover=cen5_60/>
<Add mover=relaxcart/>
<Add mover=cen5_70/>
<Add mover=relaxcart/>
<Add mover=cen5_80/>
…
```

It is important to put a relax call in between each rebuilding call, as model strain is part of the selection criteria, and the relax is necessary to properly evaluate model strain.

Example 1B: Symmetric refinement into cryoEM density

As this is a symmetric system, to correctly evaluate the energetics of the system, we need to model with symmetry-related copies present. This may be done through a two-step process: first, we run the *make_symmdef_file* script to prepare a description of the symmetry of the system in a Rosetta-readable format. Next, we enable symmetric scoring and optimization within the XML file.

The information that Rosetta needs to know about a symmetric system is encoded in the *symmetry definition file*. Its tells Rosetta: (a) how to score a structure symmetrically from only asymmetric unit interactions, and (b) how the rigid-body degrees of freedom are allowed to move to maintain the symmetry of the system.

To aid in creating a symmetry definition file from a symmetric (or near-symmetric) PDB, an application, *make_symmdef_file.pl*, has been included in src/apps/public/symmetry. To generate the symmetry definition file for TRPV1, we run the command in *scenario1_cryoem_refinement/ex_B1_make_symmdef.sh*.

```
$ROSETTA3/source/src/apps/public/symmetry/make_symmdef_file.pl \
 -m NCS -a A -i B \
 -p 3j5p_transmem.pdb -r 1000 > TRPV1.symm
```

This script needs a few pieces of information: with `-m`, the type of symmetry to generate (here NCS), with `-a`, the primary chain (here A), and with `-i`, an adjacent chain in each symmetry group, separated by spaces (here just B). For C*n* symmetries, only one adjacent chain is given; for D*n*, two are given. Finally, with `-r`, we give the contact distance between a neighbor chain and the primary chain necessary to include that subunit explicitly (here, 1000, to ensure every symmetrically related copy is included). If the input system is asymmetric, the script will make a symmetrical version of it (sometimes significantly perturbing it in the process). There are a lot of other options, including forcing symmetrical order and helical and higher-order symmetries, see the documentation!

In addition to the definition file written to STDOUT, the script will also write a file **3j5p_transmem_symm.pdb**, containing the symmetrized version of the input file, and a file **3j5p_transmem_INPUT.pdb**, that contains only the mainchain, to be used as input (in addition to the symmetry definition file).

The symmetry definition file looks something like this:

```
symmetry_name TRPV1__4
E = 4*VRT0_base + 4*(VRT0_base:VRT3_base) + 2*(VRT0_base:VRT2_base)
anchor_residue CoM
…
set_dof JUMP0_to_com x(11.7023996817515)
set_dof JUMP0_to_subunit angle_x angle_y angle_z
…
```

The omitted sections describe a system of virtual residues that maintain the symmetry of the system, and they generally should remain unedited.

The two *set_dof* lines are what the user may want to edit. There are two possibilities when refining symmetric structures into density:

A) we **don't want to refine the rigid body orientation of the entire system**;
we know the symmetry axes and we don't want them to move

B) we **do want to refine the orientation of the entire system**, including symmetry axes

Generally, in cryoEM, where the maps are symmetrically averaged, and the symmetry is known, we want to use strategy A. However, in some cases (for example, if our starting model was not perfectly symmetric) we want to use strategy B. In both cases, a minor edit to the set_dof lines in the symmdef file is necessary.

For strategy **A**, because we are using density, we need to change the first set_dof line to the following:

```
set_dof JUMP0_to_com x y z
```

For strategy **B**, we leave the two lines unchanged and instead add a third line:

```
set_dof JUMP0 x y z angle_x angle_y angle_z
```

For D*n* symmetries, the changes are similar, except in **A** the jump name is `JUMP0_0_to_com`. Alternately, for strategy **B**, the flag –d can be given to the *make_symmdef_file* script. The rest of this section uses strategy **A**; the edited symmetry definition file is in *scenario1_cryoem_refinement/TRPV1_edit.symm*

Once a symmetry definition file has been generated, then refining structures in Rosetta symmetrically is straightforward. The changes to the previous XML file are indicated below (see *scenario1_cryoem_refinement/ex_B2_symm_multicycle.xml*):

```
…
<cen weights="score4_smooth_cart" symmetric=1>
<dens_soft weights="soft_rep" symmetric=1>
<dens weights=talaris2013_cart symmetric=1>
…
<SetupForSymmetry name=setupsymm definition="./TRPV1_edit.symm"/>
…
<SymMinMover name=cenmin scorefxn=cen type=lbfgs_armijo_nonmonotone
      max_iter=200 tolerance=0.00001 bb=1 chi=1 jump=ALL/>
```

In all three declared scorefunctions, symmetric=1 must be given. Additionally, the SetupForDensityScoring mover must be replaced with the SetupForSymmetry mover. Finally, the MinMover must be replaced with its symmetric counterpart.

Example 1C: Atomic B factor fitting and reporting model-map agreement

Generally, depending on the quality of the initial models and the quality of the data, one will want to anywhere from a dozen to about 100 independent trajectories. To evaluate these models, typically, we use an independent reconstruction (that is, a reconstruction not used in model fitting) and atomic B-factor fitting of individual models into density (which provides a more accurate model of density from an atomic model).

Thus, to evaluate each of these models, we have provided XML-parsable movers to perform each of these two steps. These are illustrated in the following XML file (from *scenario1_cryoem_refinement/ex_C1_bfact_FSC.xml*):

```
<BfactorFitting name=fit_bs max_iter=50 wt_adp=0.0005 init=1 exact=1/>
<ReportFSC name=reportFSC res_low=10 res_high=3.4 nresbins=20
    testmap="TRPV1_half2.mrc"/>
```

The first mover will fit atomic B factors to maximize model-map correlation. A constraint enforcing nearby atoms to take the same B factors is also employed, and the weight on this term is controlled with the *wt_adp* term (0.0005 is generally well-behaved). Finally, *init=1* means to do a quick scan of overall B factors before beginning refinement; if there is more than one call to this mover in a single trajectory, then only the first needs to have *init=1*. *Exact=1* should almost always be used (there is a fast, approximate version, but it occasionally is poorly behaved, and uses significant amounts of memory).

The second mover will calculate a model-map FSC, and integrate over the input resolution ranges. We have found that looking at FSC is the high-resolution shells only is most informative, as in this case where we are looking only at the shells from 10Å to 3.4Å. For each call to this function, it adds a line to the header of the output PDB file:

```
REMARK   1 FSC[mask=6.75237](10:3) = 0.250239 / 0.239448
```

Here, the first number after the = sign is the agreement to the training map, and the second number is the agreement to the test map.

Because we are starting from a PDB of the tetramer, and we want Rosetta to be aware of the symmetry, we should rerun the *make_symmdef_file* script. However, since we did not refine symmetry, as a shortcut, we can simply take chain A of the output PDB, and reuse the old symmetry definition file.

Note: For the purposes of this demo, steps B and C have been split into separate steps. However, they are easily combined into a single XML file (and have been, in *scenario1_cryoem_refinement/ex_C2_full_refinement.xml*).

Example 1D: Detailed model-map agreement with density_tools
**Not available in current release*

Finally, an application, *'density_tools,'* is intended to provide additional information on model-map agreement as well as some methods for map manipulation. Some of these are briefly presented below.

To get the full FSC between a model and map using Rosetta's density model (*scenario1_cryoem_refinement/ex_D1_full_FSC.sh*):

```
bin/density_tools.linuxgccrelease \
 -database ~/Rosetta/database/ \
 -in::file::s TRPV1_INPUT_0001.pdb \
 -edensity::mapfile TRPV1_half2.mrc \
 -edensity::mapreso 3.4 \
 -edensity::cryoem_scatterers \
 -crystal_refine \
 -denstools::verbose
```

Additionally, if the arguments `-denstools::mask -denstools::mask_radius 6.0` are given, then the input model is used to mask the map. Despite the name, *mask_radius* actually controls the resolution cutoff of the mask (so with these flags, there is no contribution from the mask to the FSC above 6 Å radius). To view the calculated density (and the mask), add the flag `-denstools::dump_map_and_mask`.

To get per-residue real-space correlations between a model and a map, replace `-denstools::verbose` with:

```
-denstools::perres
```

To cut a region of density out of the map corresponding to a model, replace this flag instead with:

```
-denstools::maskonly
```

Or the inverse:

```
-denstools::cutonly
```

Scenario 2: Model rebuilding guided by experimental density data

In this scenario, we introduce a tool, RosettaCM, for building missing portions of a model guided by density data. While primarily geared towards comparative modeling, it may also be useful for building portions of a protein that are disordered when crystallized or difficult regions in hand-built models. In this scenario, we introduce the basic rebuilding protocol, then show how the tool may also be used to:

- Combine pieces from multiple template models guided by density
- Rebuild with user-defined restraints
- Iteratively rebuild models in difficult cases difficult cases

As a running example, we use the 20S proteasome (Xueming Li *et al.*, *Nature Methods*, 2013), where only a subset of particles were used, resulting in a 4.1Å reconstruction. We are building models starting from a homologous structure (pdb id: 1iru) as the starting model (25%/32% sequence identity to chains A/B).

Example 2A: Preparing templates for use in RosettaCM

In many cases, much of the setup work is handled by a script, *setup_RosettaCM.py* in RosettaTools (a separate repository available from rosettacommons.org). This script takes an input alignment in a variety of formats, and prepares the inputs automatically. It is executed by running the command:

```
setup_RosettaCM.py \
 --fasta t20s.fasta \
 --alignment tmpl.fasta \
 --alignment_format fasta \
 --templates tmpl.pdb \
 --rosetta_bin ~/Rosetta/main/source/bin \
 --verbose
```

Inputs include the full-length fasta, an alignment file – in either fasta, ClustalW, or HHSearch format – and the corresponding template PDB files. This script will prepare all the necessary inputs in order to run RosettaCM.

Alternately, the setup may be performed manually. In this case, since we are using some nonstandard features (symmetry and density) and we have two chains in the asymmetric unit we will do this; alternately, the inputs from the previous step may be used as a starting point and subsequently modified. In this case, we may convert our alignment to Rosetta format (*scenario2_model_rebuilding/20S_1iru.ali*):

```
## 1XXX_ 1iruAH_thread
# hhsearch
scores_from_program: 0 1.00
0 TVFSPDGRLFQVEYAREAVKK-GSTALGMKFANGVLLISDKKVRSRLIEQNSIEKIQLIDDYVAAVTSGLVADAR…
0 TIFSPEGRLYQVEYAFKAINQGGLTSVAVRGKDCAVIVTQKKVPDKLLDSSTVTHLFKITENIGCVMTGMTADSR…
--
```

In this format, the first line is '##' followed by a code for the target and one for the template. The second line identifies the source of the alignment; the third just keep as it is. The fourth line is the target sequence and the fifth is the template; the number is an 'offset', identifying where the sequence starts. However, the number doesn't use the PDB resid but just counts residues *starting at 0*. The sixth line is '--'. Multiple alignments may be concatenated in a single file, with the template code identifying the template corresponding to each alignment.

RosettaCM takes as inputs *partially threaded* models, that is models where aligned positions have their residue identities remapped, and unaligned residues are not present. To generate these models from an alignment file and template, we can run the Rosetta command (*scenario2_model_rebuilding/ex_A1_partialthread.sh*):

```
bin/partial_thread.linuxgccrelease \
 -database ~/Rosetta/database/ \
 -in::file::fasta t20s.fasta \
 -in::file::alignment 20S_1iru.ali \
 -in::file::template_pdb 1iruAH_aln.pdb
```

This will output a partially threaded model in *1iruA_thread.pdb* that is correctly numbered for input into RosettaCM.

Next, we need to set up symmetric modeling with RosettaCM. As in Scenario 1, we use the *make_symmdef_file.pl* script in order to generate a symmetry definition file for use in Rosetta. A straightforward way to do so is to use Chimera to dock the necessary chains into density. We need a single "primary chain" and a couple of an adjacent chain in each point group; since the proteasome features D7 symmetry, that means we need an adjacent chain in the 7-fold complex, as well as a chain in the opposite ring. An example has been created in *scenario2_model_rebuilding/setup_symm.pdb* where three copies of the threaded model have been docked into density with Chimera. To generate our D7 symmetry file from this input, we then simply have to run the command (*scenario2_model_rebuilding/ex_A2_make_symmdef.sh)*.

```
~/rosetta_source/src/apps/public/symmetry/make_symmdef_file.pl \
 -m NCS -a A -i B C \
 -p setup_symm.pdb -r 1000 > D7.symm
```

Since we have already created the input templates using the *partial_thread* application, we can ignore the *setup_symm_INPUT.pdb* file and use the output of partial thread as the input. However, we still need to align all the threaded models to this input structure. This can either be done with the program TMalign (external to Rosetta) or by using Chimera to dock the individual threaded models into density. In this case, where we have just one template, it has already been aligned to the template in *scenario2_model_rebuilding/tmpl_thread_aln.pdb.*

As in Scenario 1, we need to make a small edit to the symmetry definition file for density refinement. Change the following lines:

```
set_dof JUMP0_0_to_com x(35.3434689631743)
set_dof JUMP0_0_to_subunit angle_x angle_y angle_z
set_dof JUMP0_0 x(39.2905097135684) angle_x
```

To (*scenario2_model_rebuilding/D7_edit.symm*):

```
set_dof JUMP0_0_to_com x y z
set_dof JUMP0_0_to_subunit angle_x angle_y angle_z
```

Note: The 20S proteasome case we are using contains two chains in the asymmetric unit. To specify this as inputs to RosettaCM, we need to list the fasta file, separating the chains by the slash character '/'. This is really only necessary in the fasta provided as input to RosettaCM (next step) however, there is no harm is doing this in every step.

Example 2B: Running RosettaCM using a single template model as input.

Like the methods introduced in Scenario 1, RosettaCM is controlled through an XML script using RosettaScripts. The XML is as follows (*scenario2_model_rebuilding/ex_B1_rosettaCM_singletarget.xml)*:

```
<ROSETTASCRIPTS>
   <SCOREFXNS>
      <stage1 weights="score3" symmetric=1>
         <Reweight scoretype=atom_pair_constraint weight=0.25/>
      </stage1>
      <stage2 weights="score4_smooth_cart" symmetric=1>
         <Reweight scoretype=atom_pair_constraint weight=0.25/>
      </stage2>
      <fullatom weights="talaris2013_cart" symmetric=1>
         <Reweight scoretype=atom_pair_constraint weight=0.25/>
      </fullatom>
   </SCOREFXNS>
   <MOVERS>
      <Hybridize name=hybridize stage1_scorefxn=stage1 stage2_scorefxn=stage2
                 fa_scorefxn=fullatom batch=1>
         <Template pdb="tmpl_thread_aln.pdb" weight=1.0
                 cst_file="auto" symm_file="D7_edit.symm"/>
      </Hybridize>
   </MOVERS>
   <PROTOCOLS>
      <Add mover=hybridize/>
   </PROTOCOLS>
</ROSETTASCRIPTS>
```

The main work is done through a single mover, *Hybridize* which handles all stages of model-building. Input structures are specified via *Template* lines (in this case there is only one). For each template line, we specify the pdb input, as well as a couple of other parameters: a *weight* (the relative frequency we sample each template with); a *constraint file* (setting this to "auto" sets up automatic constraints to the template, while setting this to "none" turns off all constraints, user-defined constraints are described later); and an (optional) *symmetry definition* file.

Note: Be sure that your templates are aligned to the density!

Given this XML, RosettaCM is then run with the following command line (*scenario2_model_rebuilding/ex_B1_rosettaCM_singletarget.sh)*:

```
~/Rosetta/main/source/bin/rosetta_scripts.default.macosgccrelease \
      -database ~/Rosetta/main/database \
      -in:file:fasta t20s.fasta \
      -parser:protocol ex_B1_rosettaCM_singletarget.xml \
      -nstruct 50 \
      -edensity::mapfile t20S_41A_half1.mrc \
      -edensity::mapreso 5.0 \
      -edensity::cryoem_scatterers \
      -relax:minimize_bond_angles \
      -relax:min_type lbfgs_armijo_nonmonotone \
      -relax:jump_move true \
      -relax:default_repeats 2 \
      -default_max_cycles 200
```

The input command is similar to those seen before, but with a few key differences. First, the input to Rosetta is specified with `-in:file:fasta` rather than `-in:file:s`. Secondly, the argument `nstruct` is bolded. This argument controls the number of output structures generated. Unlike previous protocols, it is generally necessary to generate one- to several-hundred output trajectories to sufficiently sample conformational space (the exact number of structures primarily depends upon the number of unaligned residues in the input templates). Finally, the final five arguments are necessary to provide to RosettaCM for refining structures against density in the final stage.

Note (1): the XML scripts from Scenario 1 and 2 may be combined together so that every independent trajectory is subject to the full refinement protocol. This works well in practice, but may be computationally expensive, as some of the models generated by RosettaCM will be poor quality and can easily be filtered.

Note (2): for running large numbers of jobs in parallel on a cluster, the command line argument `-out:suffix` is very useful to ensure outputs do not overwrite each other.

Example 2C: Running RosettaCM using multiple template models as input.

One of the strengths of RosettaCM is its ability to make use of multiple template structures, and to recombine portions of these models during conformational sampling. This is particularly useful when multiple homologous structures are available, some with closer sequence identity, and some with more complete coverage. The protocol allows the combination of features of both models.

To make use of this feature, we simply add additional template lines in the input XML. In this case, we add the template 1ryp (*scenario2_model_rebuilding/ex_C1_rosettaCM_multitarget.xml)*:

```
…
<Hybridize name=hybridize stage1_scorefxn=stage1 stage2_scorefxn=stage2
          fa_scorefxn=fullatom batch=1>
   <Template pdb="1iruA_thread.pdb" weight=1.0
          cst_file="auto" symm_file="D7_edit.symm"/>
   <Template pdb="1rypA_thread.pdb" weight=1.0
          cst_file="auto" symm_file="D7_edit.symm"/>
</Hybridize>
…
```

The rest is handled automatically by the protocol. However, there are a few caveats when using multiple input structures:

- With density, we need to ensure that all input models are aligned to the density. This can be done using either TMalign or Chimera's alignment tools.
- In each trajectory, a starting model is chosen at random; the constraints and symmetry from this selected model are chosen at the start of each run. If we wish to use a portion of a model, but do not want to use its symmetry or constraints, we can assign it a weight of 0: backbone conformations from this model will be used in conformational sampling, but the symmetry and constraints will never be used.
- Similarly, gaps in the selected starting model are rebuilt before recombination occurs. If one of the templates has poor coverage, but provides valuable structural features, it should be used, but with weight 0.

Example 2D: Running RosettaCM with user specified constraints.

Another strength of RosettaCM is the ability to make use of additional experimental information that provides restraints over conformational space. While previously, we have used *cst_file=auto* to automatically generate constraints from template structures, if experiments provide distance constraints (or some other positional restraint, we may make use of them in model rebuilding as well.

The Rosetta documentation provides a good overview of the types of constraints that may be used, with a number of different constraint types and functional forms possible. For this demo, we will assume we have knowledge on the distance between residues 107 and 143 that we want to use during rebuilding.

This information can be encoded in a constraint file (*scenario2_model_rebuilding/ex_D1_constraints.cst)*:

```
AtomPair CA 107 CA 143 HARMONIC 5.0 1.0
```

We then replace the *cst_file=auto* lines in the XML with our own constraint file.

Note: The numbering of residues is based upon the order in the input fasta file (and does not reset between chains!).

```
…
<Template pdb="tmpl_thread_aln.pdb" weight=1.0
          cst_file="ex_D1_constraints.cst" symm_file="D7_edit.symm"/>
…
```

We can then rebuild and refine as before.

Example 2E: Model selection and running RosettaCM iteratively

With possibly hundreds of generated models, there are a few strategies to identify the best-sampled models. Generally, models should be filtered on two different criteria – the *total score* and the *density score* – in some way. We often select the best 10-20% of models based on total score, and the sort these models by density score, but visual inspection of the best by both criteria can often be beneficial in difficult cases.

Finally, one strategy for solving difficult structures is to apply RosettaCM iteratively. Using the above criteria, we can select the best 5-20 (roughly) models from the first round of refinement, and feed them as inputs into the next round. This is very briefly illustrated in the following XML (*scenario2_model_rebuilding/ex_E1_rosettaCM_iter.xml)*:

```
…
<Hybridize name=hybridize stage1_scorefxn=stage1 stage2_scorefxn=stage2
          fa_scorefxn=fullatom batch=1>
   <Template pdb="expected_outputs/S_multitgt_0001_A.pdb" weight=1.0
          cst_file="none" symm_file="D7_edit.symm"/>
   <Template pdb="expected_outputs/S_multitgt_0002_A.pdb" weight=1.0
          cst_file="none" symm_file="D7_edit.symm"/>
   <Template pdb="expected_outputs/S_multitgt_0003_A.pdb" weight=1.0
          cst_file="none" symm_file="D7_edit.symm"/>
</Hybridize>
…
```

There is also some manipulation of input models that can prove beneficial. If one wants to force rebuilding some segment of backbone, they can simply delete those residues in all input models. Similarly if one wants to force some region to adopt a conformation taking in one input model, they can delete all other conformations from all models.

Optional Scenario: Application of these tools to crystallographic data

While the previous two scenarios have focused on refinement against low-resolution EM data, we also may use low-resolution unphased crystallographic data to guide sampling as well. This section provides a very brief introduction to some of the crystallographic refinement tools. These tools are largely implemented through the RosettaScripts XML interface.

Rosetta's crystallographic scorefunction, **xtal_ml**, is implemented through calls to Phenix (Adams *et al.*, *Acta Cryst. D*, 2010) through Phenix's Python interface. Therefore, Rosetta must be compiled in a particular manner:

- Make a link of *$PHENIX_HOME/build/intel-linux-2.6-x86_64/base/lib/libpython2.7.** to *$ROSETTA3/external/lib*
- Copy *$PHENIX_HOME/build/intel-linux-2.6-x86_64/base/include/python2.7/* to *$ROSETTA3/external/include*

Then compile as usual with 'extras=python', e.g.:

```
./scons.py -j8 bin mode=release extras=python
```

Phenix contains a wrapper command, *phenix.rosetta.run_phenix_interface*, that point Rosetta and Phenix to each other. So all Rosetta commands that make use of scoring against unphased data should be called in this way:

```
phenix.rosetta.run_phenix_interface \
  ~/Rosetta/main/source/bin/rosetta_scripts.macosgccrelease \
      -parser:protocol refine.xml \
      -s model.pdb \
      -cryst::mtzfile input.mtz \
      -crystal_refine
```

The unphased crystallographic data is given (in MTZ format) with the flag `-cryst::mtzfile`. Note that this data must have only a single F/SIGF column, and must have R-free reflections defined.

Crystal symmetry may be setup using the *make_symmdef_file* script

```
~/Rosetta/src/apps/public/symmetry/make_symmdef_file.pl \
 -m CRYST \
 -p input.pdb > cryst.symm
```

Note: The input structure must have an appropriate CRYST1 line

There are also a number of movers to specifically deal with unphased crystal data. First, to setup for crystal refinement, and pass in a few options, including refinement target and twin laws:

```
    <SetRefinementOptions name=setup_opts res_high=0 res_low=0 twin_law=""
       target="ml" map_type="2mFo-DFc"/>
```

All the options given here are the default, but one can specify other refinement options.

To rephrase the data using the current model (the density map overwrites the current map, and can be accessed through the density scoring of the previous sections):

```
   <RecomputeDensityMap name=recompute_dens/>
```

To fit atomic B factors against the reciprocal space data:

```
   <FitBfactors name=fit_bs adp_strategy="individual"/>
```

To automatically set the weight on *xtal_ml* to a reasonable value (by normalizing gradients of the experimental and energetic terms):

```
<SetCrystWeight name=set_cryst_wt weight_scale=0.5 scorefxn=xtal
    scorefxn_ref=xtal cartesian=1 bb=1 chi=1/>
```

Notice that *cartesian=1* specifies that subsequent movement will be done in Cartesian space; change this to 0 if the movement is instead in torsional space.

To tag the output PDB with a line reporting R/Rfree:

```
<TagPoseWithRefinementStats name=tag tag=cycle/>
```

Additionally, refinement commonly uses two additional movers for conformational sampling. The first, *SymPackRotamersMover* performs rotamer optimization against *phased* crystal data:

```
<SymPackRotamersMover name=screpack scorefxn=dens
    task_operations=extra,restrict,keep_curr />
```

The second, *SymMinMover*, performs all-atom minimization against the unphased reflection data:

```
<SymMinMover name=min_cart_xtal cartesian=1 scorefxn=xtal
    type=lbfgs_armijo_rescored tolerance=0.0001 max_iter=100 bb=1 chi=1/>
```

Again, *cartesian=1* specifies that movement will be done in Cartesian space; change this to 0 if the movement is desired in torsional space (better-suited for low resolution and domain motions).

Finally, two common refinement protocols against crystallographic data are included in Rosetta, in *Rosetta/source/src/apps/public/crystal_refinement/:*

- *low_resolution_refine.xml* – intended for worse than 3 Å data or very distant starting structures
- *high_resolution_refine.xml* – intended for better than 3 Å data with a reasonable starting model

These protocols work well over a wide range of parameters, and so are probably sufficient for a vast majority of problems. They also serve well as starting points for custom protocols.

Scenario 3: Model rebuilding guided by experimental density data

Note: *The methods in this scenario are not yet released. They will be available in an upcoming weekly release of Rosetta. This section is included as a reference.*

This scenario presents a set of *de novo* model building tools, for building protein models into near-atomic-resolution density data when crystal structures of identifiable homologues are not available. Like Scenario 1, this tutorial uses the trans-membrane region of TRPV1 (M. Liao, E. Cao, D. Julius, Y. Cheng, *Nature*, 2013) as a running example.

This tutorial includes three folders:
scripts/
Wrappers for calling rosetta executables, setting up jobs, and results processing.

input_files/
Files necessary for running the protocol on an example case, the 3.4A TRPV1 cryo-EM map (the exact commands used in the manuscript).

denovo_model_building/
The directory where jobs will be run. Calculations are split into several steps, with each step run in an individual subfolder:

Step1_Place_fragments_into_density/
Step2.1_Calculate_overlap_scores/
Step2.2_Calculate_nonoverlap_scores/
Step3_Simulated_annealing_Monte_Carlo_sampling/

To begin, edit the configuration file, *denovo_model_building_scripts.cfg*

```
demo_dir = absolute_path_for_this_demo_in_your_work_station
```

A. Dock Fragments into density

In this step, we dock local backbone fragments into the EM density. Fragments for a particular protein may be downloaded from Robetta (www.Robetta.org). For this case, the necessary fragment files have been included.

Note: As this step is particularly time consuming, it is recommended to run on a moderately large cluster (64+ compute cores). Submission scripts for the condor queuing system are included; it should be relatively straightforward to convert these scripts for other queuing systems.

Go to the directory, *denovo_model_building/Step1_Place_fragments_into_density/condor_jobs/*, and run the command:

```
./setup_placement_condor_jobs.sh
```

This will setup a single condor job for each residue in the protein (in this example, 307 jobs). Then, to submit the jobs, simply run:

```
sh submit_placement_condor_jobs.sh
```

For running on a different system, this file will need to be updated to point to the new sequence, map, and fragment files.

After jobs finish, run a script to ensure all output has been produced. Go to the folder *denovo_model_building/Step1_Place_fragments_into_density/condor_jobs/* and run the command:

```
./find_out_unfinished_placement_jobs.sh  | grep -v input_files
```

This script will print out all residues that have not finished running (or died unexpectedly). If you need to submit one of these jobs, go to the relevant folder, remove *running.lock* and run *condor_submit placement_condor_job*. If interrupted, the code will resume from where it left off.

Once fragment placement is complete, we cluster and extract the best-scoring candidate placements. Go to the directory *denovo_model_building/Step1_Place_fragments_into_density/condor_jobs/* and run the command:

```
./setup_cluster_and_extract_condor_jobs.sh
```

This will again setup a single condor job for each residue in the protein. To launch this job, run the following command on the condor system:

```
sh submit_cluster_and_extract_condor_jobs.sh
```

Finally, once this step is complete, the output will appear in the following directory: *denovo_model_building/Step1_Place_fragments_into_density/candidate_fragment_placements/*

The output from this step is 50 candidate placements for each residue. In this example, TRPV1, it should contain 50*307=15350 placements.

Note: If the number of extracted fragments does not match, check if placement jobs haven't finished or if clustering failed, and rerun steps the corresponding job.

B**: **Precompute fragment compatibility scores

In this step, we precompute the compatibility scores of all pairs of fragments identified in the previous step. This is subdivided into two jobs, the first computes the overlap scores and the second the nonoverlap scores.

As with the previous step, this will launch a single compute job for each residue in the protein. Submission scripts for the condor queuing system are included; it should be relatively straightforward to convert these scripts for other queuing systems.

To calculate overlap scores, go to the directory *denovo_model_building/ Step2.1_Calculate_overlap_scores/* and run the command:

```
./setup_condor_jobs.sh
```

This will setup a single condor job for each residue in the protein. To launch this job, run the following command on the condor system:

```
./sh submit.sh
```

For nonoverlap scores, the setup is the same. Go to the folder *denovo_model_building /Step2.2_Calculate_nonoverlap_scores* and run the command:

```
./setup_condor_jobs.sh
```

This will setup a single condor job for each residue in the protein. To launch this job, run the following command on the condor system:

```
./sh submit.sh
```

C. Run Monte Carlo sampling to identify maximally consistent subset of fragments

In this step, different combinations of fragments are combinded to identify the maximally compatible subset of placements. These are run in many parallel Monte Carlo trajectories, in order to identify a mutually consistent subset.

First, we set up the score tables. This step simply combines the results from steps 2.1 and 2.2. Go to the directory *./denovo_model_building/Step3_Simulated_annealing_Monte_Carlo_sampling/* and run the command:

```
./setup_idx_files.sh
```

Running this script will produce four output files, which serve as inputs for the next step of the protocol:

- **frags.idx1:** an index of fragments from *candidate_fragment_placements/*
- **all_density.idx1:** the density score for each fragment
- **all_nonoverlap_scores.weighted.idx1:** closability score and clash score
- **all_overlap_scores.idx1:** overlap score

Given these inputs, we next run Monte Carlo sampling. Go to the directory *denovo_model_building/Step3_Simulated_annealing_Monte_Carlo_sampling/* and run the command:

```
./run_samc_sampling.sh [number_of_jobs]
```

This step can be run on one machine, and will launch "number_of_job" threads on the node on which it is run. This may be launched on several machines to generate additional trajectories (the outputs will not collide).

Finally, we use the results of these independent Monte Carlo trajectories in order to assemble a partial model. This will combine converged regions from the low-energy trajectories, producing a high-confidence partial model. Go to the directory *./denovo_model_building/ Step3_Simulated_annealing_Monte_Carlo_sampling/* and run the command:

```
./assemble_partial_model.sh
```

The output from this step will be in the pdb file:
denovo_model_building/Step3_Simulated_annealing_Monte_Carlo_sampling/average_model/average.pdb