UNIVERSITY of York

YSBL
York Structural Biology Laboratory

THE CLIPPER MODULE

FOR CRYSTALLOGRAPHIC COMPUTING IN PYTHON

Jon Agirre, Stuart McNicholas and Kevin Cowtan

WHO ARE WE?

Kevin Cowtan
Author of Clipper
Buccaneer, Pirate, Parrot,
Nautilus…

Stuart McNicholas
Author of CCP4mg

Jon Agirre
Author of Privateer

UNIVERSITY of York

YSBL
York Structural Biology Laboratory

THE CLIPPER C++ LIBRARY

Why use a crystallographic library?

- Because it will save you a huge amount of work:
  - 3-10x increase in productivity
  - common algorithms are already built-in
  - well designed classes prevent common coding errors
- Because it has been extensively debugged
  - by use in other programs
  - by test suites of varying degrees of formality

THE CLIPPER C++ LIBRARY

Why use a Clipper in particular?

- Purpose designed for phase improvement and interpretation, i.e. good for
  - Phasing and phase improvement
  - Model building and refinement

Why not?

- No facilities for un-merged data
- Limited facilities for anything before fixing origin

THE CLIPPER C++ LIBRARY

Originally accessible from C++ only…

- not trivial to build
- depends upon mmdb2, fftw2, libccp4, can use cctbx

…but now wrapped with SWIG

- aim: produce Python bindings
  - but bindings for other languages should be trivially generated

ONE INTERFACE, MANY WRAPPERS

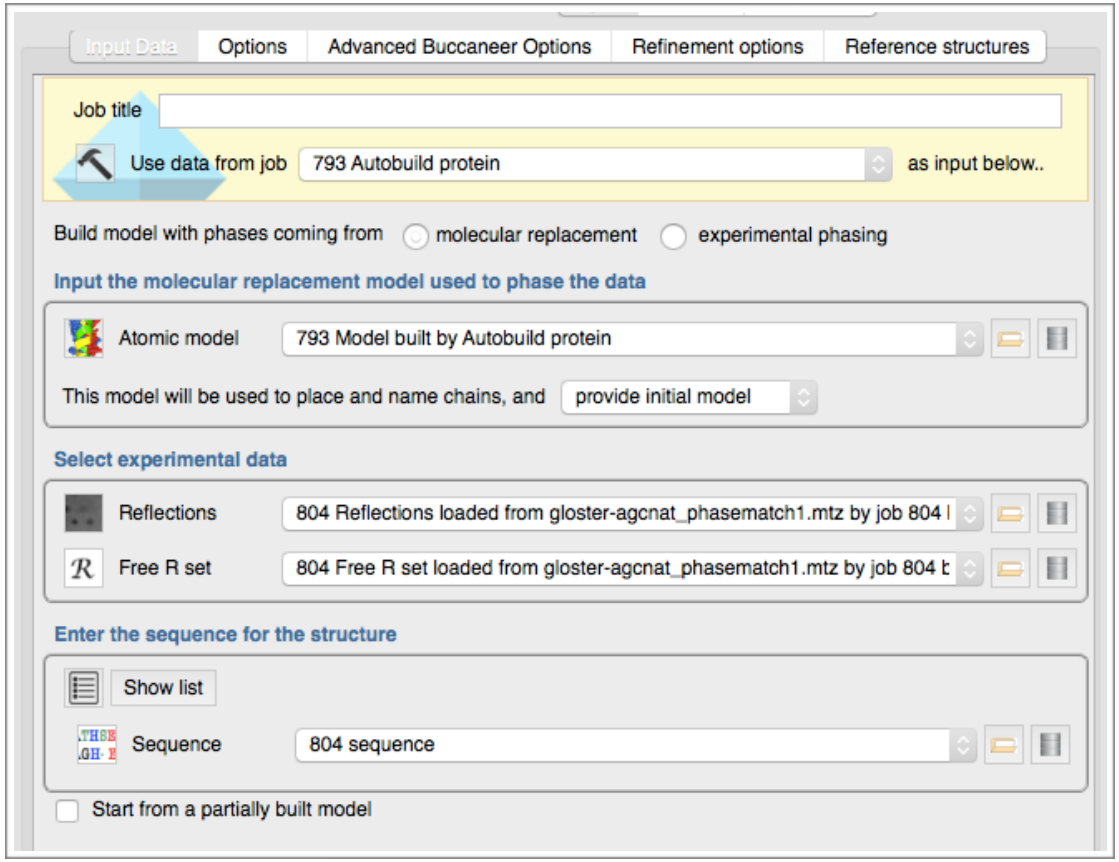
```
          ( … )

namespace std
{
   %template(UnsignedIntVector) vector<unsigned int>;
   %template(IntVector) vector<int>;
   %template(IntIntVector) vector<vector<int> >;
   %template(DoubleVector) vector<double>;
   %template(DoubleDoubleVector) vector<vector<double> >;
   %template(ClipperStringVector) vector<clipper::String>;
   %template(StringVector) vector<string>;
}

%apply std::string { clipper::String }
%apply std::string& { clipper::String& }

          ( … )
```

clipper.i

- Allegro CL
- C#
- CFFI
- CLISP
- Chicken
- D
- Go
- Guile
- Java
- Javascript
- Lua
- Modula-3
- Mzscheme
- OCAML
- Octave
- Perl
- PHP
- Python
- R
- Ruby
- Scilab
- Tcl
- UFFI

source: www.swig.org

WHY PYTHON?



CCP4 i2

Stay tuned!

HOW TO GET IT

i2 will be released later in 2015 (CCP4 7.0)

- bundled **`ccp4-python`** (> 2.7.9) and **`clipper-python`**
- for now, access through CCP4 devtools (jhbuild)
- still under development, but basic stuff works

```
bzr checkout bzr+http://oisin.rc-harwell.ac.uk/bzr/devtools/trunk devtools
cd devtools
./cj list clipper-python
./cj build cmake clipper clipper-python
./install/bin/ccp4-python
import clipper
```

HOW TO GET IT

Alpha test of CCP4 7.0 has just begun, although currently the only fully working build is the Mac one:

- http://series-70.fg.oisin.rc-harwell.ac.uk/dloads.html

You're most welcome to try it!

DOCUMENTATION

Where to go for information:

- http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html

Comprehensive *doxygen* pages covering

- class documentation
- a range of tutorials
- a few practical examples

DOCUMENTATION

Written using C++ syntax, but easy to translate

```
clipper::Xmap<float> xmap;
clipper::CCP4MAPfile f;
f.open_read( “example.map” );
f.import_xmap ( xmap );
f.close_read ();
```

C++

```
xmap = clipper.Xmap_float ()
f = clipper.CCP4MAPfile ()
f.open_read( “example.map” )
f.import_xmap_float ( xmap )
f.close_read ()
```

Python

DOCUMENTATION

Written using C++ syntax, but easy to translate

```
#include <clipper/clipper.h>
#include <clipper/clipper-cif.h>
#include <clipper/clipper-mmdb.h>
#include <clipper/clipper-ccp4.h>
#include <clipper/clipper-contrib.h>
#include <clipper/clipper-minimol.h>
#include <clipper/contrib/sfcalc_obs.h>
#include <clipper/minimol/minimol_utils.h>
```

C++

```
import clipper
```

Python

…and no complicated -lclipper-cif -lclipper-mmdb (…) lines!

BRIEF OVERVIEW OF CLASSES

OVERVIEW OF CLASSES

Crystal information, separate from actual data

- cell and symmetry

Ordinates, derivatives, operators and grids

- HKLs, coordinates, etc

Data classes

- reflection data, maps (crystallographic and otherwise)

Method objects

- common tasks, e.g. data conversion, sigmaa, etc.

Input/output classes

CRYSTAL INFORMATION

A crystal is defined by two main classes

- a unit cell (clipper.Cell)
- a spacegroup (clipper.Spacegroup)

```
dir ( clipper.Cell )
['__class__', '__del__', '__delattr__', '__dict__', '__doc__',
'__format__', '__getattr__', '__getattribute__', '__hash__',
'__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__swig_destroy__', '__swig_getmethods__',
'__swig_setmethods__', '__weakref__', '_s', 'a', 'a_star', 'alpha',
'alpha_deg', 'alpha_star', 'b', 'b_star', 'beta', 'beta_deg',
'beta_star', 'c', 'c_star', 'debug', 'equals', 'format', 'gamma',
'gamma_deg', 'gamma_star', 'init', 'is_null', 'matrix_frac',
'matrix_orth', 'metric_real', 'metric_reci', 'volume']
```

ORDINATES AND GRIDS

Ordinates include

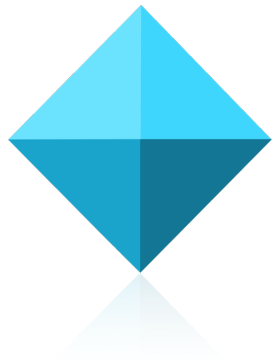
- Miller indices (clipper.HKL)
- orthogonal and fractional coordinates (clipper.Coord_orth, clipper.Coord_frac). Can be interconverted!
- grid coordinates (clipper.Coord_grid)

Operators for transforming coordinates

- rotation matrices and rotation-translation operators (e.g. clipper.RTop_orth, clipper.RTop_frac, clipper.Symop).

Gradients, curvatures, grids (clipper.Grid_sampling), etc

DATA OBJECTS



Hold the actual crystallographic data

- they include reciprocal space data (clipper.HKL_info, clipper.HKL_data),
- crystallographic and non-crystallographic maps (clipper.Xmap, clipper.NXmap) and FFT maps (clipper.FFTmap)

Design goal: hide all the bookkeeping associated with crystallographic symmetry (and in real space, cell repeat)

- Read/write operations → one copy of the data will be modified correctly. Very computationally efficient!

INPUT/OUTPUT

Objects are used to record the contents of a data object in a file or restore the contents from a file

Different objects are used for different file types, but the interfaces are as similar as the file format allows

- MTZ → CCP4MTZfile
- MAP → CCP4MAPfile
- mmCIF → MMCIFfile

INPUT/OUTPUT

dir (clipper.CCP4MTZfile)

['Default', 'Legacy', 'Normal', '__class__', '__del__', '__delattr__',
'__dict__', '__doc__', '__format__', '__getattr__',
'__getattribute__', '__hash__', '__init__', '__module__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__swig_destroy__',
'__swig_getmethods__', '__swig_setmethods__', '__weakref__',
'ccp4_spacegroup_number', 'cell', 'close_append', 'close_read',
'close_write', 'column_labels', 'column_paths', 'export_chkl_data',
'export_crystal', 'export_dataset', 'export_hkl_data',
'export_hkl_info', 'high_res_limit', 'history', 'hkl_sampling',
'import_chkl_data', 'import_crystal', 'import_dataset',
'import_hkl_data', 'import_hkl_info', 'import_hkl_list',
'low_res_limit', 'num_reflections', 'open_append', 'open_read',
'open_write', 'resolution', 'set_column_label_mode', 'set_history',
'set_spacegroup_confidence', 'set_title', 'set_verbose', 'sort_order',
'spacegroup', 'spacegroup_confidence', 'title']

INPUT/OUTPUT

Reading structure factors in two popular formats

```
cif_file = clipper.CIFfile()
hkl_info = clipper.HKL_info()
cif_file.open_read ( “test.cif” )
cif_file.import_hkl_info ( hkl_info )
sg, cell = hkl_info.spacegroup(), hkl_info.cell()

mtz_file = clipper.CCP4MTZfile()
hkl_info = clipper.HKL_info()
mtz_file.open_read ( “test.mtz” )
mtz_file.import_hkl_info ( hkl_info )
sg, cell = hkl_info.spacegroup(), hkl_info.cell()
```

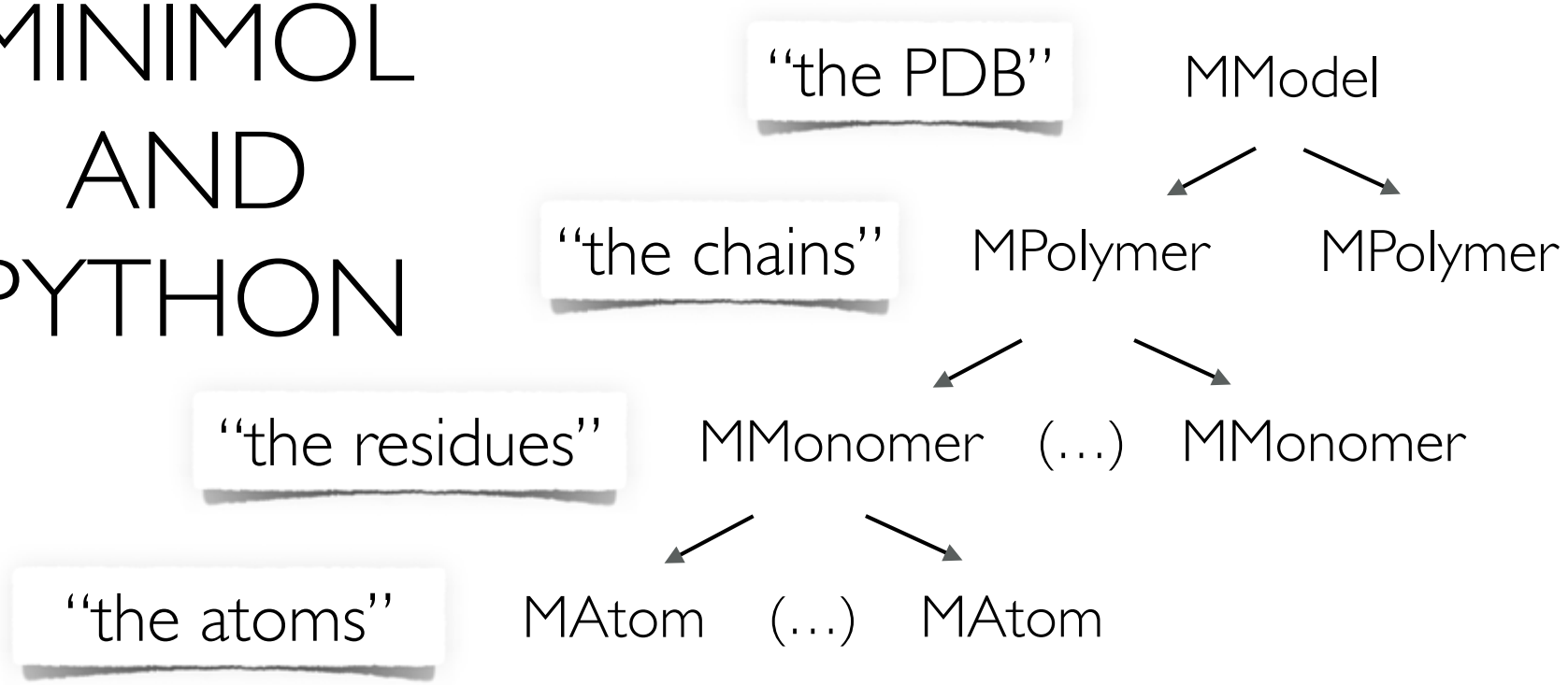
HANDLING ATOMIC MODELS

We've got two options…

- clipper.MMDB package: a minimal interface to MMDB to allow clipper Cell, Spacegroup, and Atom_list objects to be initialised from MMDB objects
- clipper::MiniMol package: MMDB is powerful but complex. MiniMol can be used for simple tasks, is easy to learn and hard to break

…but MiniMol works beautifully with Python

MINIMOL AND PYTHON




```
f = clipper.MMDBfile()
f.read_file ( "test.pdb")
mmol = clipper.MiniMol ()
f.import_minimol ( mmol )

for polymer in mmol.model () :
  for monomer in polymer :
    for atom in monomer :
      print atom.coord_orth().x(),atom.coord_orth().y(),atom.coord_orth().z()
```

HANDY STUFF

Tons of basic stuff in clipper.Util

```
dir (clipper.Util)
['__class__', '__del__', '__delattr__', '__dict__', '__doc__',
'__format__', '__getattr__', '__getattribute__', '__hash__',
'__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__swig_destroy__', '__swig_getmethods__',
'__swig_setmethods__', '__weakref__', 'atanh', 'b2u', 'bessel_i0',
'd2rad', 'eightpi2', 'intc', 'intf', 'intr', 'invsim', 'is_nan',
'is_null', 'isnan', 'mod', 'nan', 'nand', 'nanf', 'pi', 'rad2d',
'set_null', 'sim', 'sim_deriv', 'sim_deriv_recur', 'sim_integ',
'twopi', 'twopi2', 'u2b']
```

MORE DETAILS AND EXAMPLES

CRYSTAL INFORMATION

We almost never make objects from scratch, but we can using the compact descriptions

```
cell_desc = clipper.Cell_descr ( 30.0, 40.0, 50.0 )
the_cell = clipper.Cell ( cell_desc )
sg_desc = clipper.Spgr_descr ( 1 )
the_sg = clipper.Spacegroup ( sg_desc )
```

Cell description takes up to 6 arguments (a, b, c, alpha, beta, gamma), degrees or radians

Spacegroup description can be H-M or Hall symbol, spacegroup number, or list of operators

CRYSTAL INFORMATION

Accessing associated information:

```
print the_cell.a()
30.0
print the_cell.volume()
60000.0
print the_cell.a_star()
0.0333333333333
print the_cell.matrix_orth()
<clipper.Mat33_float; proxy of <Swig Object of type 'clipper::Mat33<
float > *' at 0x10d8b4f60> >
print the_cell.matrix_orth()[0][0]
30.0
print the_sg.num_primitive_symops()
1
print the_sg.symop(0)
<clipper.Symop; proxy of <Swig Object of type 'clipper::Symop *' at
0x10d8b4c00> >
```

COORDINATES

Build orthogonal coordinates from *float* triplets

```
coords = clipper.Coord_orth ( 10.0, 20.0, 30.0 )
print coords.x()
10.0
```

Rotate & translate coordinates using sg symops

```
coords_symop0 = coords.transform ( the_sg.symop(0).rtop_orth(the_cell) )
coords_symop1 = coords.transform ( the_sg.symop(1).rtop_orth(the_cell) )
coords_symop2 = coords.transform ( the_sg.symop(2).rtop_orth(the_cell) )
```

COORDINATES

Example transformations

```
coord_grid = clipper.Coord_grid ( 1, 2, 3 )
print coord_grid.format()
uvw = (   1,   2,   3)

grid = clipper.Grid_sampling ( 100, 200, 300 )
coord_frac = coord_grid.coord_frac ( grid )
print coord_frac.format()
uvw = (      0.01,      0.01,      0.01)

coord_frac = coord_frac * the_sg.symop ( 0 )
print the_cell.format()
Cell (    30,    40,    50,    90,    90,    90)

coord_orth = coord_frac.coord_orth ( the_cell )
print coord_orth.format ( )
xyz = (       0.3,       0.4,       0.5)
```

HKL DATA

Create from integers

```
hkl = clipper.HKL ( 1, 2, 3 )
```

Calculate resolution

```
print hkl.invresolsq ( the_cell )
0.00721111111111
```

Calculate phase shift

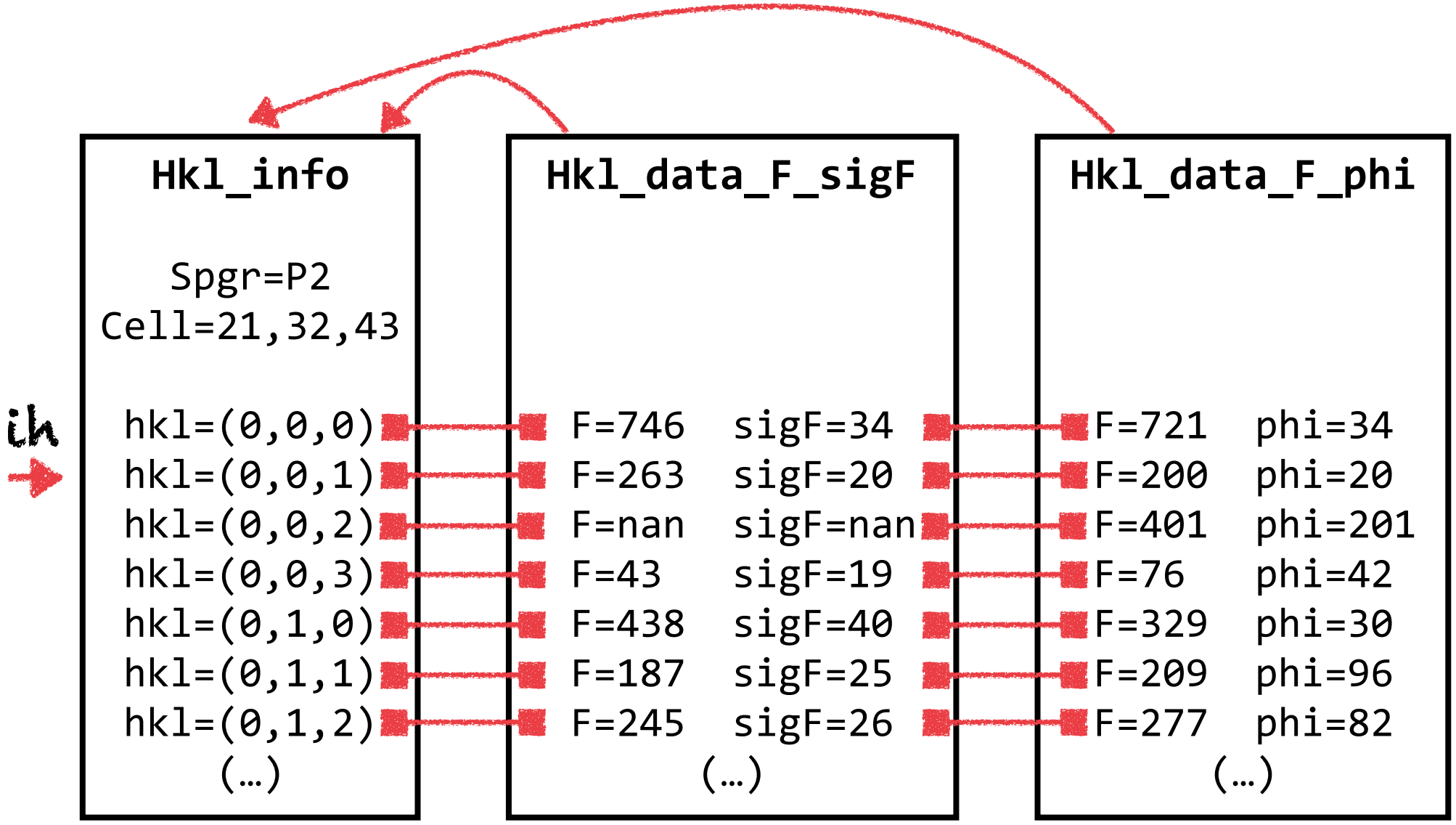
```
print hkl.sym_phase_shift ( the_sg.symop ( 0 ) )
-0.0
```

RECIPROCAL SPACE DATA

We often want to handle many lists of related reflection data, handling all the data connected with one HKL at once

Clipper implements a system of data lists, holding data of crystallographic types, using a common indexing defined by a parent object (HKL_reference_index)

RECIPROCAL SPACE DATA



"book keeping"

ih = HKL_reference_index

RECIPROCAL SPACE DATA

Some available classes:

- HKL_data_F_sigF_float (and double)
- HKL_data_I_sigI_float (and double)
- HKL_data_F_Phi_float (and double)
- HKL_data_ABCD_float
- HKL_data_Phi_fom_float
- HKL_data_Flag

RECIPROCAL SPACE DATA

Equivalent HKL_data sets may share HKL_info and HKL_reference_index objects

- for instance when accessing F_sigF and FreeR sets

```
ih = HKL_data_F_Phi_float.HKL_reference_index()

for ih in range ( len ( other_flag ) ):
  if ( not f_sigf[ih].missing() )
  and ( freerset[ih].missing() or freerset[ih].flag() == 1 ) :
    other_flag[ih].set_flag ( clipper.SFweight_spline_float.BOTH )
  else :
    other_flag[ih].set_flag ( clipper.SFweight_spline_float.NONE )
```

MAPS

Two classes for real space maps: Xmap and NXmap (makes sense for EM data, for instance)

Reading a map from disk

```
xmap = clipper.Xmap_float ()
map_file = clipper.CCP4MAPfile ()
map_file.open_read ( “example.map”)
map_file.import_xmap_float ( xmap )
map_file.close_read ()
sg, samp, cell =  f.spacegroup(), f.grid_sampling(), f.cell()
```

Maps can be looped over using Map_reference_index

METHOD OBJECTS

Resolution functions – used to calculate smoothly varying estimates of things in reciprocal space, *e.g.* |F(|h|)|

Electron density/structure factor calculation from atoms

Map filtering, e.g. for calculating local mean of variance of electron density

Skeletonisation

Likelihood weighting and map calculation

METHOD OBJECTS

Example 1: structure factor calculation

```
atoms  = mmol.atom_list()
f_sigf = clipper.HKL_data_F_sigF_float(…)
f_calc = clipper.HKL_data_F_phi_float(…)
sf_calculator = clipper.SFcalc_obs_bulk_float()
sf_calculator ( f_calc, f_sigf, atoms )
```

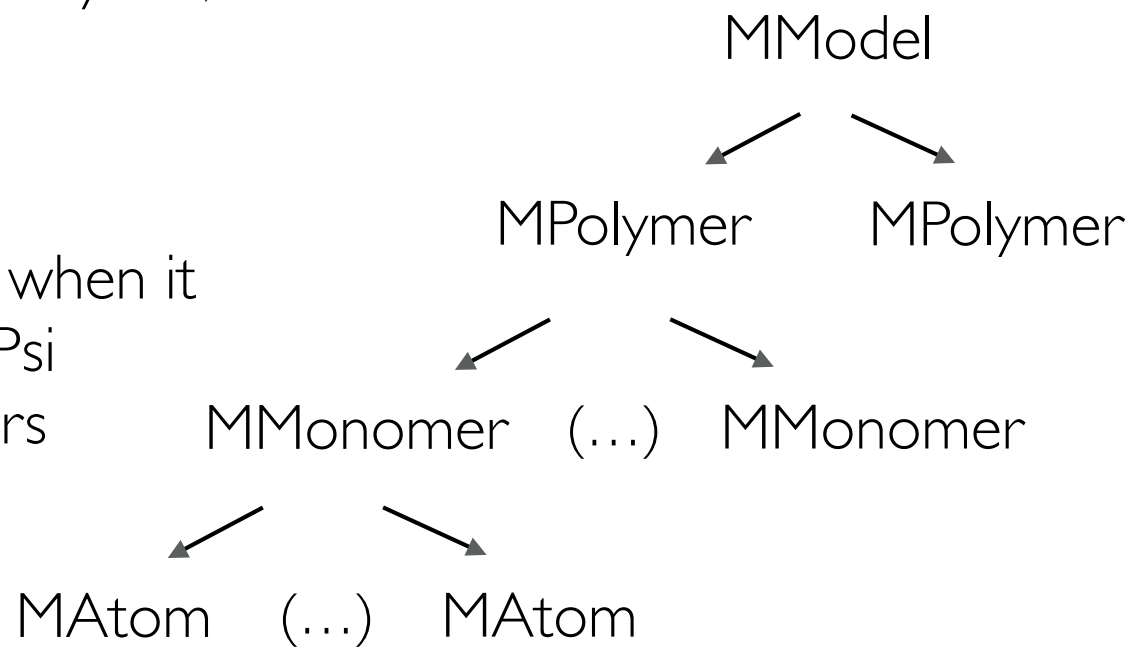
Example 2: weighted map calculation

```
best_coefficients = clipper.HKL_data_F_phi_float(…)
difference_coefficients = clipper.HKL_data_F_phi_float(…)
phi_weighted = clipper.HKL_data_Phi_fom_float(…)
freerflag = clipper.HKL_data_Flag(…)
calculator = clipper.SFweight_spline_float ( 1000, 20 )
calculator ( best_coefficients, difference_coefficients,
             phi_weighted, f_sigf, f_calc, freerflag )
```

WORKING WITH ATOMIC MODELS

Vertical operations: get atom list, find, select, lookup, insert, copy operations available within MModel, MPolymer, MMonomer

Horizontal operations also when it makes sense, *e.g.* get Phi/Psi between two MMonomers



Plus primitive operations on MAtom as expected: set_element, set_coord_orth, set_occupancy, set_u_iso…

WORKING WITH ATOMIC MODELS

All objects can be transformed by an RTop_orth

Built-in Dunbrack's and Richardsons' rotamer libraries (within MMonomer). Also Richardsons' Top500-based Ramachandran tables (get angles with MMonomer, access tables within clipper.Ramachandran)

- this is what Coot (Emsley et al., 2010) uses

PLAY TIME IDEAS

A geometry validation program?

```
mon1 = clipper.MMonomer (…)
mon2 = clipper.MMonomer (…)

phi = clipper.MMonomer.protein_ramachandran_phi ( mon1, mon2 )
psi = clipper.MMonomer.protein_ramachandran_phi ( mon1, mon2 )

rama_gly = clipper.Ramachandran ( clipper.Ramachandran.Gly )

print rama_gly.allowed ( phi, psi )
True

print rama_gly.allowed ( phi - 10, psi )
False

print rama_gly.favored ( phi - 10, psi )
False
```

OTHER IDEAS

Load a map and get statistics

Generate whole unit cell using sg symops

Extend data to lower symmetry space group

Calculate weighted maps from PDB + F_sigF

CONCLUSIONS

Clipper contains a great many building blocks for the rapid construction of crystallographic calculations, now also in Python

It is suitable for an increasing range of problems

Convenience methods provide simple ways to do complex tasks

Binary core under scripting layer → good performance

Lots of documentation and examples, with more to come embedded in CCP4 i2!

WORK IN PROGRESS!

PLEASE REPORT ANY BUGS TO

JON.AGIRRE@YORK.AC.UK
STUART.MCNICHOLAS@YORK.AC.UK
KEVIN.COWTAN@YORK.AC.UK

- feature requests more than welcome too -