Will's PyMOL Scripts Documentation

Release 0.2

Will Sheffler

Mar 12, 2017

Contents

CHAPTER 1

Basics

Intro

3D Geom functions (not dependent on pymol) and lots of pymol-specific stuff

You *should* be able to run / import these file without changing your path

To test, run 'python -m unittest discover' in the pymol directory. To test an individual module, run that file and it will do all the doctests.

To build docs, you need Sphinx (I use v1.1.3), which is the standard python doc generator cd into doc and 'make html'. You can add docs as you like in the docstrings and in doc/source.

1Tutorial

Compatibility with pyrosetta Vectors/Matrices

use v.to_rosetta() and Vec(rosetta_vec)

to/from rosetta.numeric.xyzVector_double_t

```
>>> import rosetta
>>> vros = rosetta.numeric.xyzVector_double_t(1,2,3)
>>> # convert from rosetta xyzVector to Vec
>>> v = Vec(vros)
>>> print(type(v))
<class 'xyzMath.Vec'>
>>> print(v)
(1.000000,2.000000,3.000000)
>>> # convert to rosetta xyzVector from Vec
>>> u = v.to_rosetta()
```

```
>>> print(type(u))
<class 'rosetta.numeric.xyzVector_double_t'>
>>> print(u)
      1.000000000000000       2.000000000000000       3.000000000000000
```

to/from rosetta.numeric.xyzMatrix

```
>>> import rosetta
>>> mros = rosetta.numeric.xyzMatrix_double_t(0)
>>> mros.xx(1); mros.yy(1); mros.zz(1)
>>> # convert from rosetta xyzMatrix to Mat
>>> m = Mat(mros)
>>> print(type(m))
<class 'xyzMath.Mat'>
>>> print(m)
Mat[ (1.000000,0.000000,0.000000), (0.000000,1.000000,0.000000), (0.000000,0.000000,1.
↪000000) ]
>>> # convert to rosetta xyzMatrix from Mat
>>> n = m.to_rosetta()
>>> print(type(n))
<class 'rosetta.numeric.xyzMatrix_double_t'>
>>> print(n)
      1.000000000000000       0.000000000000000       0.000000000000000
      0.000000000000000       1.000000000000000       0.000000000000000
      0.000000000000000       0.000000000000000       1.000000000000000
```

Getting helical parameters from a transform

Get the transform you are interested in

first, get the transform you're interested in. if you are in pymol and want to get a transform between selection, you can do this:

```
from pymol_util import getrelframe
xform = getrelframe( 'resi 1 and name n+ca+c', 'resi 4 and name n+ca+c' )
```

if you have atomic coordinates (say from pyrosetta) and want to use those to define stubs, you can do something like this

```
>>> some_xform = rotation_around_degrees(axs=Vec(1,0,0), ang=180.0, cen=Vec(0,1,0))
>>> some_xform.t.x = 10 # translation along axis
>>> N_1  = Vec(1,0,0)
>>> CA_1 = Vec(0,1,0)
>>> C_1  = Vec(0,0,1)
>>> N_2  = some_xform * N_1
>>> CA_2 = some_xform * CA_1
>>> C_2  = some_xform * C_1
>>> print N_2, CA_2, C_2
(11.000000,2.000000,-0.000000) (10.000000,1.000000,0.000000) (10.000000,2.000000,-1.
↪000000)
```

then use the stub function do get stubs (coordinate frames) from the coords:

```
>>> stub1 = stub(N_1, CA_1, C_1)
>>> stub2 = stub(N_2, CA_2, C_2)
```

now the transform that takes stub1 to stub2 is the following (approx equal to some_xform in this example):

```
>>> xform = stub2 * ~stub1
>>> assert xform == some_xform
```

Get the helical parameters

axis of rotation, rotation magnitude and center of rotation can be computed with the *rotation_axis_center* method of class *Xform*

```
>>> axis, ang, cen = xform.rotation_axis_center()
```

translation along the rotation axis can be obtained:

```
>>> translation_along_axis_of_rotation = axis.dot(xform.t)
>>> print translation_along_axis_of_rotation
10.0
```

helical radius is relative to the coordinates. can be obtained with *projperp()*

Note: rounding is only so these code examples test correctly

```
>>> radius_from_N_1  = projperp(axis, cen - N_1 ).length()
>>> radius_from_CA_1 = projperp(axis, cen - CA_1).length()
>>> radius_from_C_1  = projperp(axis, cen - C_1 ).length()
>>> round(radius_from_N_1, 6)
1.0
>>> round(radius_from_CA_1, 6)
0.0
>>> round(radius_from_C_1, 6)
1.414214
```

sanity check: transform shouldn't change the radius

```
>>> radius_from_N_2  = projperp(axis, cen - N_2 ).length()
>>> radius_from_CA_2 = projperp(axis, cen - CA_2).length()
>>> radius_from_C_2  = projperp(axis, cen - C_2 ).length()
>>> assert round(radius_from_N_1 , 6) == round(radius_from_N_2 , 6)
>>> assert round(radius_from_CA_1, 6) == round(radius_from_CA_2, 6)
>>> assert round(radius_from_C_1 , 6) == round(radius_from_C_2 , 6)
```

CHAPTER 2

Module Documentation

3D Geometry Library (not pymol specific)

Points, Vectors, Lines, Planes, Matrices, Rigid Transforms with associated functions Easy 3D Linear Algebra, like xyz* in rosetta

xyzMath.**isint**(*x*)

xyzMath.**isfloat**(*x*)

xyzMath.**isnum**(*x*)

xyzMath.**isiter**(*x*)

xyzMath.**islist**(*x*)

xyzMath.**istuple**(*x*)

xyzMath.**isvec**(*x*)

xyzMath.**ismat**(*x*)

xyzMath.**isxform**(*x*)

class xyzMath.**Vec**(*x=0.0*, *y=None*, *z=None*)

a Vector like xyzVector<Real> in rosetta

```
>>> v = Vec(1,2,3)
>>> print v, 10*v
(1.000000,2.000000,3.000000) (10.000000,20.000000,30.000000)
```

multiplication is a dot prod at the moment >>> v*v 14.0 >>> assert Vec(1,0,-0) == Vec(1,-0,0)

to_rosetta(*v*)

dot(*u*, *v*)

normdot(*u*, *v*)

angle(*u*, *v*)

angle_degrees(*u*, *v*)

lineangle(*u*, *v*)

lineangle_degrees(*u*, *v*)

length(*u*)

length_squared(*u*)

distance(*u*, *v*)

distance_squared(*u*, *v*)

cross(*u*, *v*)

normalize(*u*)

normalized(*u*)

outer(*u*, *v*)

rounded(*sd*)

unit(*v*)

abs(*v*)

tuple(*v*)

key(*v*)

round0(*v*)

xyzMath.**randvec**(*n=None*)

xyzMath.**randveccube**(*r=1.0*)

xyzMath.**randvecball**(*r=1.0*)

xyzMath.**randnorm**(*n=None*)

```
>>> assert abs(randnorm().length()-1.0) < 0.0000001
```

xyzMath.**coplanar**(*x1*, *x2*, *x3*, *x4*)

```
>>> u,v,w = randvec(3)
>>> a,b,c = (gauss(0,10) for i in range(3))
>>> assert     coplanar(u, v, w, u + a*(u-v) + b*(v-w) + c*(w-u) )
>>> assert not coplanar(u, v, w, u + a*(u-v) + b*(v-w) + c*(w-u) + randvec().
↪cross(u-v) )
```

xyzMath.**rmsd**(*l*, *m*)

```
>>> l,m = randvec(6),randvec(6)
>>> rmsd(l,l)
0.0
```

class xyzMath.**Mat**(*xx=None*, *xy=None*, *xz=None*, *yx=None*, *yy=None*, *yz=None*, *zx=None*, *zy=None*, *zz=None*)

docstring for Mat

```
>>> m = Mat(2,0,0,0,1,0,0,0,1)
>>> print m
Mat[ (2.000000,0.000000,0.000000), (0.000000,1.000000,0.000000), (0.000000,0.
↪000000,1.000000) ]
>>> print m*m
Mat[ (4.000000,0.000000,0.000000), (0.000000,1.000000,0.000000), (0.000000,0.
↪000000,1.000000) ]
>>> print Mat(*range(1,10)) * Mat(*range(10,19))
Mat[ (84.000000,90.000000,96.000000), (201.000000,216.000000,231.000000), (318.
↪000000,342.000000,366.000000) ]
>>> assert Mat(0.0,1.0,2.0,3,4,5,6,7,8) == Mat(-0,1,2,3,4,5.0,6.0,7.0,8.0)
>>> print Mat(100,2,3,4,5,6,7,8,9).det()
-297.0
>>> m = Mat(100,2,3,4,5,6,7,8,9)
>>> assert m * ~m == Imat
```

to_rosetta(*m*)

row(*m*, *i*)

col(*m*, *i*)

rowx(*m*)

rowy(*m*)

rowz(*m*)

colx(*m*)

coly(*m*)

colz(*m*)

transpose(*m*)

transposed(*m*)

det(*m*)

trace(*m*)

add_diagonal(*m*, *v*)

is_rotation(*m*)

rotation_axis(*R*)

```
>>> axis ,ang  = randnorm(),uniform(-pi,pi)
>>> axis2,ang2 = rotation_matrix(axis,ang).rotation_axis()
>>> assert abs( abs(ang) - abs(ang2) ) < EPS
>>> assert axis == axis2 * copysign(1,ang*ang2)
```

euler_angles()

from_euler_angles(*euler*)

xyzMath.**projection_matrix**(*v*)

xyzMath.**proj**(*u*, *v*)

```
>>> u = Vec(1,0,0); v = Vec(1,1,1)
>>> proj(u,v)
Vec( 1.000000, 0.000000, 0.000000 )
```

xyzMath.**projperp**(*u*, *v*)

```
>>> u = Vec(1,0,0); v = Vec(1,1,1)
>>> projperp(u,v)
Vec( 0.000000, 1.000000, 1.000000 )
```

xyzMath.**rotation_matrix**(*axis*, *angle*)

xyzMath.**rotation_matrix_degrees**(*axis*, *angle*)

get a rotation matrix

```
>>> rx180 = rotation_matrix_degrees(Vec(1,0,0),180.0)
>>> rx90  = rotation_matrix_degrees(Vec(1,0,0),90.0)
>>> print rx90*rx90 == rx180
True
>>> r = rotation_matrix_degrees(Vec(1,0,0),45.0)
>>> print r
Mat[ (1.000000,0.000000,0.000000), (0.000000,0.707107,-0.707107), (0.000000,0.
↪707107,0.707107) ]
>>> assert r*r == rx90
>>> assert r*r*r*r == rx180
>>> assert r*r*r*r*r*r*r*r == Imat
>>> assert ~r == r.transposed()
```

```
>>> ang = uniform(0,1)*360.0-180.0
>>> v = randvec()
>>> axs = randnorm()
>>> while(abs(v.dot(axs))>0.9): axs = randnorm()
>>> u = rotation_matrix_degrees(projperp(v,axs),ang)*v
>>> assert abs(u.angle_degrees(v)-abs(ang)) < SQRTEPS
>>> test_rotation_mat()
test_rotation_mat PASS
```

xyzMath.**test_rotation_mat**()

xyzMath.**randrot**(*n=None*)

class xyzMath.**Xform**(*R=None*, *t=None*)

Coordinate frame like rosetta Xform, behaves also as a rosetta Stub

```
>>> x = Xform(R=Imat,t=Uz)
>>> print x
Xform( Mat[ (1.000000,0.000000,0.000000), (0.000000,1.000000,0.000000), (0.000000,
↪0.000000,1.000000) ], (0.000000,0.000000,1.000000) )
>>> assert (x*x) == Xform(R=Imat,t=2*Uz)
>>> x = Xform(R=rotation_matrix_degrees(Vec(1,0,0),90.0),t=Vec(0,0,0))
>>> print x
Xform( Mat[ (1.000000,0.000000,0.000000), (0.000000,0.000000,-1.000000), (0.
↪000000,1.000000,0.000000) ], (0.000000,0.000000,0.000000) )
>>> assert x*x*x*x == Ixform
>>> x.t = Ux
>>> assert x*x*x*x == Xform(R=Imat,t=4*Ux)
>>> x.t = Uz
```

```
>>> print x
Xform( Mat[ (1.000000,0.000000,0.000000), (0.000000,0.000000,-1.000000), (0.
↪000000,1.000000,0.000000) ], (0.000000,0.000000,1.000000) )
>>> assert x               == Xform(R=rotation_matrix_degrees(Ux, 90.0),t=Vec(0,␣
↪0,1))
>>> assert x*x             == Xform(R=rotation_matrix_degrees(Ux,180.0),t=Vec(0,-
↪1,1))
>>> assert x*x*x           == Xform(R=rotation_matrix_degrees(Ux,270.0),t=Vec(0,-
↪1,0))
>>> assert x*x*x*x         == Xform(R=rotation_matrix_degrees(Ux,  0.0),t=Vec(0,␣
↪0,0))
>>> assert x*x*x*x*x       == Xform(R=rotation_matrix_degrees(Ux, 90.0),t=Vec(0,␣
↪0,1))
>>> assert x*x*x*x*x*x     == Xform(R=rotation_matrix_degrees(Ux,180.0),t=Vec(0,-
↪1,1))
>>> assert x*x*x*x*x*x*x   == Xform(R=rotation_matrix_degrees(Ux,270.0),t=Vec(0,-
↪1,0))
>>> assert x*x*x*x*x*x*x*x == Xform(R=rotation_matrix_degrees(Ux,  0.0),t=Vec(0,␣
↪0,0))
>>> x = Xform(rotation_matrix_degrees(Vec(1,2,3),123),Vec(5,7,9))
>>> assert ~x *  x == Ixform
>>> assert  x * ~x == Ixform
```

Frames / RTs are interchangable:

```
>>> fr = Xform(rotation_matrix_degrees(Vec(1,2,3), 65.64),t=Vec(3,2,1))
>>> to = Xform(rotation_matrix_degrees(Vec(7,5,3),105.44),t=Vec(10,9,8))
>>> x = to/fr
>>> assert to/Ixform ==  to
>>> assert Ixform/fr == ~fr
>>> assert (to * ~fr) * fr == to
>>> assert x * fr == to
```

```
>>> a1 = randnorm()
>>> b1 = randnorm()
>>> ang = uniform(0,1)*360.0-180.0
>>> a2 = rotation_matrix_degrees(a1.cross(randnorm()),ang) * a1
>>> b2 = rotation_matrix_degrees(b1.cross(randnorm()),ang) * b1
>>> assert abs(angle(a1,a2) - angle(b1,b2)) < EPS
>>> xa = Xform().from_two_vecs(a1,a2)
>>> xb = Xform().from_two_vecs(b1,b2)
>>> assert xa.tolocal(a1) == xb.tolocal(b1)
>>> assert xa.tolocal(a2) == xb.tolocal(b2)
>>> assert ~xa*a1 == ~xb*b1
>>> assert ~xa*a2 == ~xb*b2
>>> assert xb/xa*a1 == b1
>>> assert xb/xa*a2 == b2
```

add/sub with Vecs:

```
>>> X = randxform()
>>> u,v = randvec(2)
>>> assert isxform(u+X) and isxform(X+u) and isxform(u-X) and isxform(X-u)
>>> assert X*v+u == (u+X)*v
>>> assert X*(v+u) == (X+u)*v
>>> assert Xform(u)*X*v == (u+X)*v
>>> assert X*Xform(u)*v == (X+u)*v
>>> assert X*v-u == (u-X)*v
```

```
>>> assert X*(v-u) == (X-u)*v
```

mul,div with Mats:

```
>>> R = randrot()
>>> assert isxform(R*X) and isxform(X*R)
>>> assert R*X*u == (R*X)*u == R*(X*u)
>>> assert X*R*u == (X*R)*u == X*(R*u)
>>> assert Xform(R)*X*u == Xform(R)*(X*u)
>>> assert X*Xform(R)*u == X*(Xform(R,V0)*u)
>>> assert X/X*v == v
```

mul/div Xforms:

```
>>> Y = randxform()
>>> assert isxform(X/Y) and isxform(X*Y)
>>> assert X/Y*v == X*~Y*v
```

these don't work yet:

```
>>> axis,ang,cen = randnorm(),uniform(-pi,pi),randvec()
>>> X = rotation_around(axis,ang,cen)
>>> axis2,ang2,cen2 = X.rotation_center()
>>> assert abs( abs(ang) - abs(ang2) ) < EPS
>>> assert axis == axis2 * copysign(1,ang*ang2)
>>> print cen
>>> print cen2
```

```
>>> x =                 Xform( Mat( Vec( 0.816587, -0.306018, 0.489427 ), Vec( 0.
↪245040, 0.951487, 0.186086 ), Vec( -0.522629, -0.032026, 0.851959 ) ), Vec( 1.
↪689794, 1.535762, -0.964428 ) )
>>> assert repr(x) == "Xform( Mat( Vec( 0.816587, -0.306018, 0.489427 ), Vec( 0.
↪245040, 0.951487, 0.186086 ), Vec( -0.522629, -0.032026, 0.851959 ) ), Vec( 1.
↪689794, 1.535762, -0.964428 ) )"
```

from_four_points(*s*, *cen*, *a*, *b*, *c*)

from_two_vecs(*s*, *a*, *b*)

tolocal(*s*, *x*)

toglobal(*s*, *x*)

inverse()

rotation_axis(*X*)

rotation_axis_center(*X*)

pretty()

xyzMath.**read_tokens**(*f*)

xyzMath.**read_xforms**(*fname*, *N=9000000000.0*, *start=0*)

xyzMath.**stub**(*cen=None*, *a=None*, *b=None*, *c=None*)

xyzMath.**randxform**(*n=None*)

xyzMath.**rotation_around**(*axs*, *ang*, *cen=None*)

```
>>> x = rotation_around(Ux,1,Uy)
>>> x * Uy
Vec( 0.000000, 1.000000, 0.000000 )
```

xyzMath.**rotation_around_degrees**(*axs*, *ang*, *cen=None*)

xyzMath.**RAD**(*axs*, *ang*, *cen=None*)

xyzMath.**test**()

xyzMath.**dihedral**(*p1*, *p2*, *p3*, *p4*)

```
>>> dihedral_degrees(Ux,Uy,V0,Uz)
90.0
>>> dihedral_degrees(Ux,V0,Uy,Uz)
-90.0
```

xyzMath.**dihedral_degrees**(*p1*, *p2*, *p3*, *p4*)

xyzMath.**angle**(*p1*, *p2*, *p3=None*)

xyzMath.**angle_degrees**(*p1*, *p2*, *p3=None*)

xyzMath.**sin_cos_range**(*x*)

xyzMath.**point_line_distance**(*p*, *a*, *c*)

```
>>> point_line_distance(V0,Uy,V0)
0.0
>>> round(point_line_distance(V0,Uy,Ux+Uz),8)
1.41421356
>>> round(point_line_distance(Ux,Ux,Vec(3,2,1)) , 8)
2.23606798
```

xyzMath.**line_line_angle**(*a1*, *a2*)

xyzMath.**line_line_angle_degrees**(*a1*, *a2*)

xyzMath.**line_line_distance**(*a1*, *c1*, *a2*, *c2*)

```
>>> line_line_distance(Ux,V0,Uy,V0)
0.0
>>> round(line_line_distance(Ux,Vec(0,1,2),Ux,Vec(3,2,1)) , 8)
1.41421356
>>> line_line_distance(Ux,10*Uy,Uz,99.0*Ux)
10.0
>>> X = randxform()
>>> round(line_line_distance(X.R*Ux,X*Vec(0,1,2),X.R*Ux,X*Vec(3,2,1)) , 8)
1.41421356
```

xyzMath.**line_line_closest_points**(*A1*, *C1*, *A2*, *C2*)

```
>>> print line_line_closest_points(Ux,Ux,Uy,Uy)
(Vec( 0.000000, 0.000000, 0.000000 ), Vec( 0.000000, 0.000000, 0.000000 ))
```

```
>>> print line_line_closest_points(Ux,Uy,Uy,Uz)
(Vec( 0.000000, 1.000000, 0.000000 ), Vec( 0.000000, 1.000000, 1.000000 ))
```

xyzMath.**skew_lines_center**(*A1*, *C1*, *A2*, *C2*)

```
>>> skew_lines_center(Ux,V0,Uy,V0)
Vec( 0.000000, 0.000000, 0.000000 )
```

```
>>> skew_lines_center(Ux,Uy,Uy,Ux)
Vec( 1.000000, 1.000000, 0.000000 )
```

```
>>> skew_lines_center(10*Ux,10*Uy,10*Uy,10*Uz)
Vec( 0.000000, 10.000000, 5.000000 )
```

>>> skew_lines_center(Ux,Uy,Uz,Ux)

xyzMath.**skew_lines_relation**(*a1*, *c1*, *a2*, *c2*)

xyzMath.**skew_lines_relation_z**(*axis*, *cen*)

xyzMath.**align_skew_line_pairs**(*aa1*, *ac1*, *aa2*, *ac2*, *ba1*, *bc1*, *ba2*, *bc2*)

```
>>> aa1 = Ux
>>> ac1 = V0
>>> aa2 = Uy
>>> ac2 = V0
>>> ba1 = Ux
>>> bc1 = V0
>>> ba2 = Uy
>>> bc2 = V0
>>> print align_skew_line_pairs(aa1,ac1,aa2,ac2,ba1,bc1,ba2,bc2)
Xform( Mat[ (1.000000,-0.000000,-0.000000), (-0.000000,1.000000,0.000000), (0.
↪000000,-0.000000,1.000000) ], (0.000000,0.000000,0.000000) )
```

```
>>> aa1 = Uy
>>> ac1 = V0
>>> aa2 = Uz
>>> ac2 = V0
>>> ba1 = Ux
>>> bc1 = Uz
>>> ba2 = Uy
>>> bc2 = Uz
>>> print align_skew_line_pairs(aa1,ac1,aa2,ac2,ba1,bc1,ba2,bc2)
Xform( Mat[ (0.000000,1.000000,0.000000), (0.000000,0.000000,1.000000), (1.000000,
↪0.000000,0.000000) ], (0.000000,0.000000,1.000000) )
```

xyzMath.**sindeg**(*x*)

xyzMath.**cosdeg**(*x*)

xyzMath.**asindeg**(*x*)

xyzMath.**acosdeg**(*x*)

xyzMath.**rotation_around_dof_to_set_vec_vec_angle**(*dofaxis*, *tgt0*, *v1*, *v2*)

```
>>> from random import random
>>> for i in range(10):
...         dof = randnorm()
...         tgt = random()*180
...         v1 = randnorm()
...         v2 = randnorm()
...         ANGs = rotation_around_dof_to_set_vec_vec_angle(dof,tgt,v1,v2)
...         for a in ANGs:
...                 R = rotation_around_degrees(dof,a,Vec(0,0,0))
...                 act = line_line_angle_degrees(v1,R*v2)
...                 if 0.000001 < min(abs(act-tgt),abs(act-180+tgt)):
...                         print a,tgt,act
>>> print "if no other output, correct"
if no other output, correct
```

xyzMath.**ray_sphere_intersection**(*lin*, *l0in*, *cen*, *r*)

```
>>> v = randnorm()
>>> v = Ux
>>> assert v.distance( ray_sphere_intersection(v,V0,V0,1) ) < 0.00001
>>> assert not ray_sphere_intersection(v,V0,-2*v,1.0)
```

xyzMath.**line_plane_intersection**(*l*, *l0*, *n*, *p0*)

```
>>> l  = Ux
>>> l0 = randvec()
>>> n  = Ux
>>> p0 = V0
>>> assert line_plane_intersection(l,l0,n,p0)[1] == Vec(0,l0.y,l0.z)
>>> n = randnorm()
>>> p0 = randvec().cross(n)
>>> l = randvec()
>>> l0 = p0+l*gauss(0,10)
>>> assert line_plane_intersection(l,l0,n,p0)[1] == p0
```

xyzMath.**slide_to_make_lines_intersect**(*dof*, *l*, *l0*, *m*, *m0*)

```
>>> v = randvec()
>>> assert abs(slide_to_make_lines_intersect(Ux,Uy,v,Uz,V0) + v.x ) < EPS
>>> dof,l,l0,m,m0 = randvec(5)
>>> d = slide_to_make_lines_intersect(dof,l,l0,m,m0)
>>> l0 = l0 + d*dof
>>> assert abs(line_line_distance(l,l0,m,m0)) < EPS
```

xyzMath.**alignvector**(*a*, *b*)

```
>>> u = randvec()
>>> v = randvec()
>>> assert v.angle(alignvector(u,v)*u) < EPS
```

xyzMath.**alignaroundaxis**(*axis*, *u*, *v*)

```
>>> axis = randnorm()
>>> u = randvec()
>>> ang = uniform(-pi,pi)
>>> v = rotation_matrix(axis,ang)*u
>>> uprime = alignaroundaxis(axis,u,v)*u
>>> assert v.angle(uprime) < EPS
>>> v = randvec()
>>> uprime = alignaroundaxis(axis,u,v)*u
>>> assert coplanar(V0,axis,v,uprime)
```

xyzMath.**alignvectors_minangle**(*a1*, *a2*, *b1*, *b2*)

exact alignment:

```
>>> for i in range(10):
...         angdeg = uniform(-180,180)
...         a1 = randvec()
...         b1 = randnorm()*a1.length()
...         l2 = gauss(0,1)
...         a2 = rotation_matrix_degrees(a1.cross(randnorm()),angdeg) * a1 * l2
...         b2 = rotation_matrix_degrees(b1.cross(randnorm()),angdeg) * b1 * l2
...         assert abs(angle(a1,a2) - angle(b1,b2)) < EPS
...         Xa2b = alignvectors_minangle(a1,a2,b1,b2)
...         assert Xa2b.t.length() < EPS
...         assert (Xa2b*a1).distance(b1) < EPS
...         assert (Xa2b*a2).distance(b2) < EPS
```

if angle(a1,a2) != angle(b1,2b), minimize deviation

```
>>> a1,a2,b1,b2 = randvec(4)
>>> Xa2b = alignvectors_minangle(a1,a2,b1,b2)
>>> assert coplanar(b1,b2,Xa2b*a1,Xa2b*a2)
>>> assert (b1.angle(a1)+b2.angle(a2)) > (b1.angle(Xa2b*a1)+b2.angle(Xa2b*a2))
```

>>> tgt1 = -Vec(0.816497,0.000000,0.577350) # >>> tgt2 = Vec(0.000000,0.000000,1.000000) # >>> orig1 = Vec(0.000000,0.000000,1.000000) # >>> orig2 = Vec(-0.723746,0.377967,-0.577350) # >>> print orig1.angle_degrees(orig2) # >>> print tgt1.angle_degrees(tgt2) # >>> x = alignvectors_minangle(orig1,orig2,tgt1,tgt2) # >>> print tgt1,x*orig1 # >>> print tgt2,x*orig2

xyzMath.**alignvectors**(*a1*, *a2*, *b1*, *b2*)

same as alignvectors_minangle

xyzMath.**get_test_generators1**()

xyzMath.**expand_xforms**(*G*, *N=3*, *c=Vec(1.000000, 3.000000, 10.000000)*, *maxrad=9000000000.0*)

```
>>> G = get_test_generators1()
>>> for x in expand_xforms(G): print x*Ux
(-1.000000,0.000000,0.000000)
(1.000000,0.000000,0.000000)
(1.000000,-0.000000,0.000000)
(-1.000000,0.000000,0.000000)
(1.000000,-2.000000,0.000000)
(1.000000,0.000000,0.000000)
(-1.000000,2.000000,0.000000)
(-1.000000,0.000000,0.000000)
(1.000000,-2.000000,0.000000)
(1.000000,-0.000000,-2.000000)
```

xyzMath.**find_identities**(*G*, *n=6*, *c=Vec(1.000000, 3.000000, 10.000000)*)

```
>>> G = get_test_generators1()
>>> for I in find_identities(G): print I.t
(0.000000,0.000000,0.000000)
(-2.000000,2.000000,2.000000)
(2.000000,-2.000000,2.000000)
```

xyzMath.**get_cell_bounds_orthogonal_only**(*G*, *n=6*, *c=Vec(1.000000, 3.000000, 10.000000)*)

very slow... need to speed up

```
>>> G = get_test_generators1()
>>> get_cell_bounds_orthogonal_only(G[:2],12)
(4.0, 4.0, 4.0)
```

xyzMath.**cyclic_axis**(*coords*)

xyzMath.**symmetrize_xform**(*anchor*, *x*, *nf=None*)

>>> x = rotation_around_degrees(Uz,180,V0) # >>> assert symmetrize_xform(Ux,x,2) == x #>>> x = rotation_around_degrees(Vec(0,0,1),121,Vec(1,1,1)) #>>> x,c = symmetrize_xform(Ux,x,3); print x.pretty(); print c # >>> print x.pretty()

General Pymol Utilities

show axes, transform things, etc.

pymol_util.**inpymol**()

pymol_util.**rainbow_chains**()

pymol_util.**showaxes**()

class pymol_util.**PutCenterCallback**(*name*, *corner=0*)

prev_v = None

load()

pymol_util.**corneraxes**(*name='axes'*)

DESCRIPTION

Puts coordinate axes to the lower left corner of the viewport.

pymol_util.**showaxes2**()

pymol_util.**getchain**(*sele*)

pymol_util.**getres**(*sele*, *withchain=True*)

pymol_util.**getrestypes**(*sele*)

pymol_util.**com**(*sel='all'*, *state=1*)

pymol_util.**showcom**(*sel='all'*)

pymol_util.**cgo_sphere**(*c*, *r=1*, *col=(1, 1, 1)*)

pymol_util.**showsphere**(*c*, *r=1*, *col=(1, 1, 1)*, *lbl=''*)

pymol_util.**showvecfrompoint**(*a*, *c*, *col=(1, 1, 1)*, *lbl=''*)

pymol_util.**cgo_segment**(*c1*, *c2*, *col=(1, 1, 1)*)

pymol_util.**showsegment**(*c1*, *c2*, *col=(1, 1, 1)*, *lbl=''*)

pymol_util.**cgo_cyl**(*c1*, *c2*, *r*, *col=(1, 1, 1)*, *col2=None*)

pymol_util.**showcyl**(*c1*, *c2*, *r*, *col=(1, 1, 1)*, *col2=None*, *lbl=''*)

pymol_util.**showline**(*a*, *c*, *col=(1, 1, 1)*, *lbl=''*)

pymol_util.**cgo_lineabs**(*a*, *c*, *col=(1, 1, 1)*)

pymol_util.**showlineabs**(*a*, *c*, *col=(1, 1, 1)*, *lbl=''*)

class pymol_util.**ResBB**(*n*, *ca=None*, *c=None*, *ss=None*)

docstring for ResBB

rms(*s*, *o*)

stub(*r*)

class pymol_util.**DisulfLib**(*fn*)

docstring for DisulfLib

disulf_rms(*r1*, *r2*)

pymol_util.**chirality**(*fe1*, *fe2*, *fe3*, *fe4*)

pymol_util.**bond_zns**(*sel*)

pymol_util.**trans**(*sel*, *v*)

pymol_util.**transx**(*sel*, *x*)

pymol_util.**transy**(*sel*, *y*)

pymol_util.**transz**(*sel*, *z*)

pymol_util.**rot**(*sel*, *axis*, *ang*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**rotx**(*sel*, *ang*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**roty**(*sel*, *ang*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**rotz**(*sel*, *ang*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**xform**(*sel*, *x*)

pymol_util.**rot_by_matrix**(*sel*, *R*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**rotrad**(*sel*, *axis*, *ang*, *cen=None*)

pymol_util.**test**(*x*, *y*, *z*)

pymol_util.**rotview**(*axis*, *ang*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**pointaxis**(*sel*)

pymol_util.**alignaxis**(*sel*, *newaxis*, *oldaxis=None*, *cen=Vec(0.000000, 0.000000, 0.000000)*)

pymol_util.**mysetview**(*look=Vec(0.000000, 0.000000, 1.000000)*, *up=Vec(0.000000, 1.000000, 0.000000)*, *pos=None*, *cen=None*, *ncp=None*, *fcp=None*)

pymol_util.**meancoords**(*sel1*, *sel2*, *n='mix'*, *w=0.5*)

pymol_util.**mygetview**()

pymol_util.**swell**()

pymol_util.**mkhelix**(*sel*, *t*, *r*, *n*)

pymol_util.**mkhelix4**(*sel*, *t*, *r*, *n*)

pymol_util.**mirror**(*sel*, *nname='mirror'*, *crd=0*)

pymol_util.**inversion**(*sel*, *nname='inv'*)

pymol_util.**mkc4**(*sel*, *a=Vec(1.000000*, *0.000000*, *0.000000)*, *c=Vec(0.000000*, *0.000000*, *0.000000)*)

pymol_util.**mkc3**(*sel*, *a=Vec(0.000000*, *0.000000*, *1.000000)*, *c=Vec(0.000000*, *0.000000*, *0.000000)*)

pymol_util.**mkc2**(*sel*, *a=Vec(0.000000*, *0.000000*, *1.000000)*, *c=Vec(0.000000*, *0.000000*, *0.000000)*)

pymol_util.**mkd2**(*sel='all'*)

pymol_util.**alignbb**(*sel='all'*, *obj=None*)

pymol_util.**alignall**(*sel='all'*, *obj=None*)

pymol_util.**fitall**(*sel='all'*, *obj=None*)

pymol_util.**centerall**(*sel='all'*)

pymol_util.**showcst**(*fname*)

pymol_util.**bondzn**()

pymol_util.**loadmov**(*d*)

pymol_util.**drawlines**(*p*, *d*, *lab='lines'*, *COL=(1*, *1*, *1)*, *SIZE=20.0*)

pymol_util.**drawtestconeva**(*v*, *a*)

pymol_util.**conelineinter**(*p*, *d*, *v*, *a*, *t*)

pymol_util.**createpoint**(*sele*, *p*, *lab*)

pymol_util.**test_conelineinter**(*sele*)

pymol_util.**useRosettaRadii**()

pymol_util.**expandRadii**(*delta=1.0*, *sel='all'*)

pymol_util.**contractRadii**(*delta=1.0*, *sel='all'*)

pymol_util.**useOccColors**(*sel='all'*)

pymol_util.**useTempColors**(*sel='all'*)

pymol_util.**useOccRadii**(*sel='all'*)

pymol_util.**useTempRadii**(*sel='all'*)

pymol_util.**natom**(*sel='all'*)

pymol_util.**nca**(*sel='all'*)

pymol_util.**chaincount**(*sel='all'*)

pymol_util.**getname1**(*s*)

pymol_util.**lcs**(*S*, *T*)

pymol_util.**sort_dcoms**(*subs*)

pymol_util.**procD5dat**(*lfile=None*, *biod='/data/biounit'*, *outd=None*)

pymol_util.**untangle_sidechains**(*sele*)

pymol_util.**orb_cyl**(*lab=''*)

pymol_util.**drawring**(*p1=None, p2=None, p3=None, col=[1, 1, 1], lab='ring'*)

pymol_util.**drawringcar**(*c, a, r, col=[1, 1, 1], lab='ring'*)

pymol_util.**drawsph**(*col=[1, 1, 1], lab='sph'*)

pymol_util.**dsf**(*CA1*, *CB1*, *CA2*, *CB2*, *lab=''*)

pymol_util.**alignallrms**(*sele*)

pymol_util.**mkpntx**(*s1*, *s2*)

pymol_util.**ifsphab**()

pymol_util.**charge**(*sel*)

pymol_util.**redoA**(*sel='not sub'*, *N=None*)

pymol_util.**redotoi**(*sel='not sub*'*)

pymol_util.**redopent**(*sel*)

pymol_util.**getaa**(*c*, *r*, *o='all'*)

pymol_util.**mkifaceresfile**(*fn=None*)

pymol_util.**color_by_chain**()

pymol_util.**getnative**()

pymol_util.**floats2vecs**(*i*)

pymol_util.**nnb**(*v*, *s*, *r*)

pymol_util.**testhsphere**(*rratio=2.0*)

pymol_util.**testsphere**()

pymol_util.**stubalign**(*s='all'*, *s1='pk1'*, *s2='pk2'*, *s3='pk3'*)

pymol_util.**tmpdoit**()

pymol_util.**symmetrizec2**(*sel1*, *sel2*)

pymol_util.**corresponding_atom_names**(*sel1*, *sel2*, *file*)

pymol_util.**bestalign**(*s1*, *s2*)

pymol_util.**process_native**()

pymol_util.**move_up_down_add_to_ignore_list**(*sel*)

pymol_util.**my_get_obj**(*enabled_only=0*)

pymol_util.**move_down**()

pymol_util.**move_up**()

pymol_util.**cbow**(*sel='all'*)

pymol_util.**print_chains**(*sele='all'*)

pymol_util.**renumber**(*selection='all'*, *start=1*, *startsele=None*, *quiet=1*)

DESCRIPTION

Set residue numbering (resi) based on connectivity.

ARGUMENTS

selection = string: atom selection to renumber {default: all}

start = integer: counting start {default: 1}

startsele = string: residue to start counting from {default: first in selection}

pymol_util.**color_obj**(*rainbow=0*)

AUTHOR

Gareth Stockwell

USAGE

color_obj(rainbow=0)

This function colours each object currently in the PyMOL heirarchy with a different colour. Colours used are either the 22 named colours used by PyMOL (in which case the 23rd object, if it exists, gets the same colour as the first), or are the colours of the rainbow

SEE ALSO

util.color_objs()

pymol_util.**tcdock_set_chain_by_cluster**()

pymol_util.**hsv_to_rgb**(*hsv*)

pymol_util.**make_zdock_set**(*d='/work/sheffler/Dropbox/project/zdock/pdb_lib'*, *tgt='/work/sheffler/data/zdock_AB'*)

pymol_util.**make_inputs_from_cb_only**(*d='/work/sheffler/Dropbox/test/silva/run_resl_6/'*, *tgt='/work/sheffler/tmp/asym_iface'*)

pymol_util.**get_closest_atom_pair**(*selpairs_or_sel1*, *sel2=None*)

pymol_util.**get_first_last_resi**(*sele*)

pymol_util.**cube**(*lb=Vec(-10.000000, -10.000000, -10.000000)*, *ub=Vec(10.000000, 10.000000, 10.000000)*, *r=0.5*, *xform=Xform(Mat(Vec(1.000000, 0.000000, 0.000000)*, *Vec(0.000000, 1.000000, 0.000000)*, *Vec(0.000000, 0.000000, 1.000000))*, *Vec(0.000000, 0.000000, 0.000000))*)

pymol_util.**getframe**(*obj*)

pymol_util.**getrelframe**(*newobj*, *refobj*, *Forigin=None*)

get transform between two objects, assume the obj's are identical

pymol_util.**getrelframe_rmsalign**(*movsel*, *refsel*, *Forigin=None*)

get transform between two objects using rmsalign

pymol_util.**tmpvis**(*s*)

pymol_util.**show_res_frames**(*sele*)

Symmetric Modeling Utilities

get axes and align at stuff

sym_util.**get_xforms_by_chain**(*sele='all'*, *verbose=False*, *userms=False*)

sym_util.**find_symelems**(*sele_or_xforms='all'*, *verbose=False*)

sym_util.**guessdxaxes**(*sele='all'*, *verbose=False*)

sym_util.**alignndx**(*sele='all'*, *verbose=False*)

sym_util.**guesscxaxis**(*sele*, *nfold=None*, *chains0=[]*, *extrasel='name CA'*)

sym_util.**aligncx**(*sele*, *nfold*, *alignsele=None*, *tgtaxis=Vec(0.000000, 0.000000, 1.000000)*, *chains=[]*, *extrasel='name CA'*)

sym_util.**align_helix**(*sele*, *nrepeat*, *tgt_axis=Vec(1.000000, 0.000000, 0.000000)*)

sym_util.**symmetrize**(*sele='not symmetrized_*'*, *alignsele=None*, *chains=[]*, *delete=True*)

sym_util.**showcxaxis**(*sele*, *nfold=None*, *chains=[]*, *length=30*, *col=(1, 1, 1)*, *lbl='Cx Axis'*)

sym_util.**myint**(*s*)

sym_util.**selbycomp**(*trn=0*)

sym_util.**rechain**(*sel*, *nres*)

sym_util.**makekinwire**(*sel*, *movres*, *fixres*)

sym_util.**get_contigs**(*x*, *n=7*)

```
>>> test = list(range(1,8)) + list(range(20,33)) + list(range(40,44)) +␣
↪list(range(49,50))+ list(range(0,8))
>>> print test
[1, 2, 3, 4, 5, 6, 7, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 40, 41,␣
↪42, 43, 49, 0, 1, 2, 3, 4, 5, 6, 7]
```

```
>>> print get_contigs( test )
[[1, 2, 3, 4, 5, 6, 7], [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32], [0,␣
↪1, 2, 3, 4, 5, 6, 7]]
```

sym_util.**get_fixed_size_contigs**(*x*, *n=7*)

```
>>> test = list(range(1,8)) + list(range(20,33)) + list(range(40,44)) +␣
↪list(range(49,50))+ list(range(0,8))
>>> print test
[1, 2, 3, 4, 5, 6, 7, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 40, 41,␣
↪42, 43, 49, 0, 1, 2, 3, 4, 5, 6, 7]
```

```
>>> for f in get_fixed_size_contigs(test,7): print f
[1, 2, 3, 4, 5, 6, 7]
[20, 21, 22, 23, 24, 25, 26]
[21, 22, 23, 24, 25, 26, 27]
[22, 23, 24, 25, 26, 27, 28]
[23, 24, 25, 26, 27, 28, 29]
[24, 25, 26, 27, 28, 29, 30]
[25, 26, 27, 28, 29, 30, 31]
[26, 27, 28, 29, 30, 31, 32]
[0, 1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
```

```
>>> for f in get_fixed_size_contigs(test,9): print f
[20, 21, 22, 23, 24, 25, 26, 27, 28]
[21, 22, 23, 24, 25, 26, 27, 28, 29]
[22, 23, 24, 25, 26, 27, 28, 29, 30]
[23, 24, 25, 26, 27, 28, 29, 30, 31]
[24, 25, 26, 27, 28, 29, 30, 31, 32]
```

```
>>> print len(get_fixed_size_contigs(test,1))
28
```

```
>>> for f in get_fixed_size_contigs(test,4): print f
[1, 2, 3, 4]
[2, 3, 4, 5]
[3, 4, 5, 6]
[4, 5, 6, 7]
[20, 21, 22, 23]
[21, 22, 23, 24]
[22, 23, 24, 25]
[23, 24, 25, 26]
[24, 25, 26, 27]
[25, 26, 27, 28]
[26, 27, 28, 29]
[27, 28, 29, 30]
[28, 29, 30, 31]
[29, 30, 31, 32]
[0, 1, 2, 3]
[1, 2, 3, 4]
[2, 3, 4, 5]
[3, 4, 5, 6]
[4, 5, 6, 7]
```

sym_util.**tmpname**()

sym_util.**gen_helical_alignments**(*sele1*, *sele2*, *pref='HALN'*)

sym_util.**colorI53**(*sel='visible'*)

sym_util.**alignsym**(*sel='all'*, *arch='I32'*, *ax1=Vec(0.000000, 0.000000, 1.000000)*, *ax2=Vec(0.356825, 0.000002, 0.934171)*)

sym_util.**xtal_frames**(*tgt=None*, *skip=()*, *r=100*)

sym_util.**make_helix_old**(*sele='vis'*, *n=30*, *nfold=1*)

sym_util.**color_by_2component**(*col1='green'*, *col2='cyan'*)

sym_util.**make_ab_components**(*dir*)

sym_util.**trim_sym**(*sel='all'*, *na=1*, *nb=1*)

sym_util.**nulltest**()

```
>>> print "foo"
foo
```

sym_util.**load_tests**(*loader*, *tests*, *ignore*)

Symmetric Component Generation Utilities

symmetric component library generation stuff, kinda obsolete at the moment

sym_comp.**homogenizechains**(*sel='all'*)

sym_comp.**processhomomers**()

sym_comp.**iscontig**(*sel*)

sym_comp.**procCdat**(*N=3*, *lfile=None*, *biod='/data/biounit'*, *outd=None*)

sym_comp.**procD2dat**(*lfile=None*, *biod='/data/biounit'*, *outd=None*)

sym_comp.**prepare_c2_nmr**(*pattern*, *outdir=None*)

sym_comp.**make_cryst1_P432**(*fn*)

sym_comp.**make_cryst1_I213**(*fn*)

sym_comp.**make_cryst1_23**(*fn*, *a2in*, *i2*, *a3in*, *i3*)

sym_comp.**nulltest**()

```
>>> print "foo"
foo
```

sym_comp.**load_tests**(*loader*, *tests*, *ignore*)

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

s

x

Index

F

G

H

I

K

L

M

N

U

V

X