MATH:7450 (22M:305) Topics in Topology: Scientific and Engineering Applications of Algebraic Topology

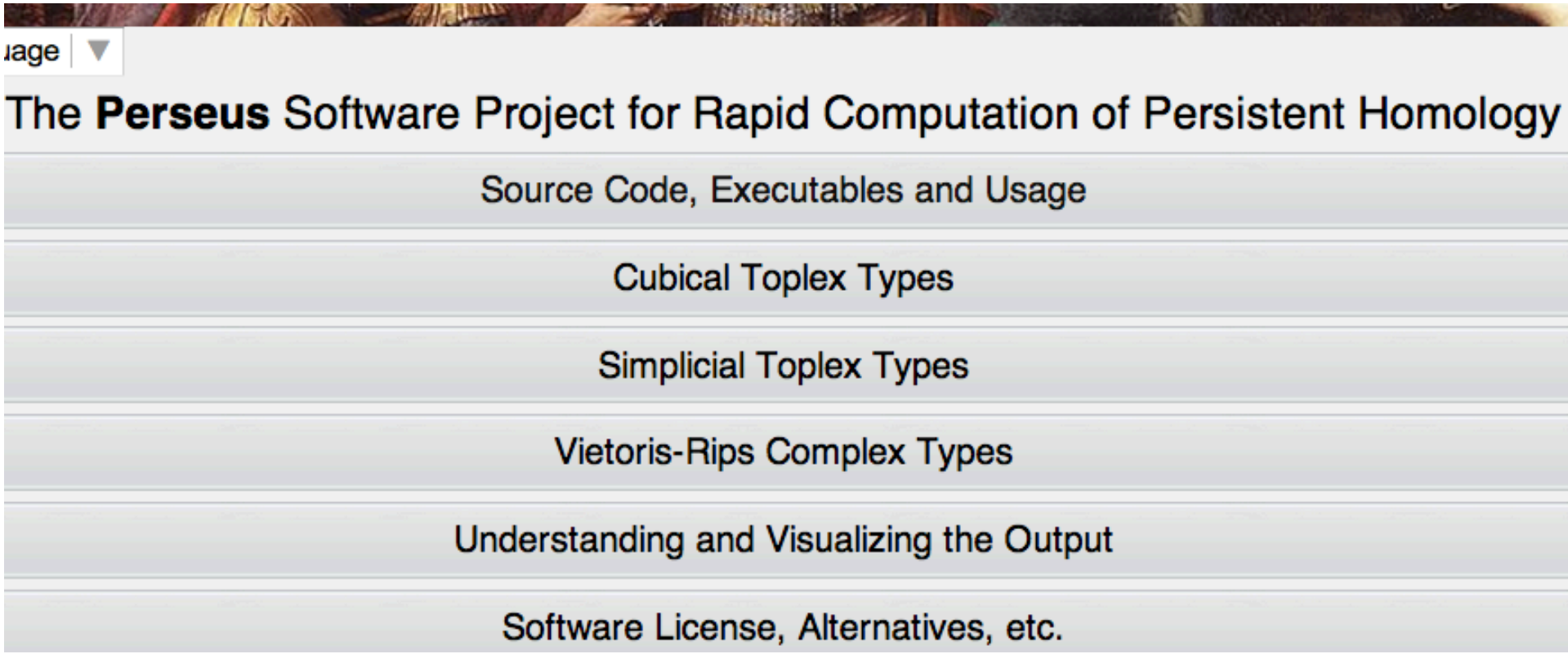
Oct 18, 2013: Dionysus (and a brief intro to Perseus)

Fall 2013 course offered through the University of Iowa Division of Continuing Education

Isabel K. Darcy, Department of Mathematics
Applied Mathematical and Computational Sciences, University of Iowa

http://www.math.uiowa.edu/~idarcy/AppliedTopology.html

Now that you have a C compiler, you can download and install a variety of software packages. For example:



http://www.math.rutgers.edu/~vidit/perseus/

Click on Here to download

The **Perseus** Software Project for Rapid Computation of Persistent Homology

Source Code, Executables and Usage

Here is a zipped file containing the C++ source code. The current version is: **3.0 Beta**. There is no need to downlo
specialized add-on such as the excellent boost libraries to use Perseus since everything is written in plain vanilla C-
utilizes only the standard template library. You can either compile executables directly from the source code provi
download the appropriate platform-dependent executable given below.

Compiling from Source

Download and extract the zip file from the link above to a directory where you have read/write permissions. You can now
Use any C++ compiler to compile the main file **Pers.cpp**. The choice of compiler depends mainly on your operating s
Microsoft Windows users have various compiler options such as the open-source mingw, or the complete inte
development environment provided by the somewhat pricey Microsoft Visual Studio. Mac users will probably require
Xcode download and installation on their systems.

Instructions for using the gcc compiler from the command line are extremely simple. Just go to the directory with the ex
source files and type:

- g++ Pers.cpp -O3 -o perseus

Of course, you can replace "perseus" in the command above with any executable name of your choice.

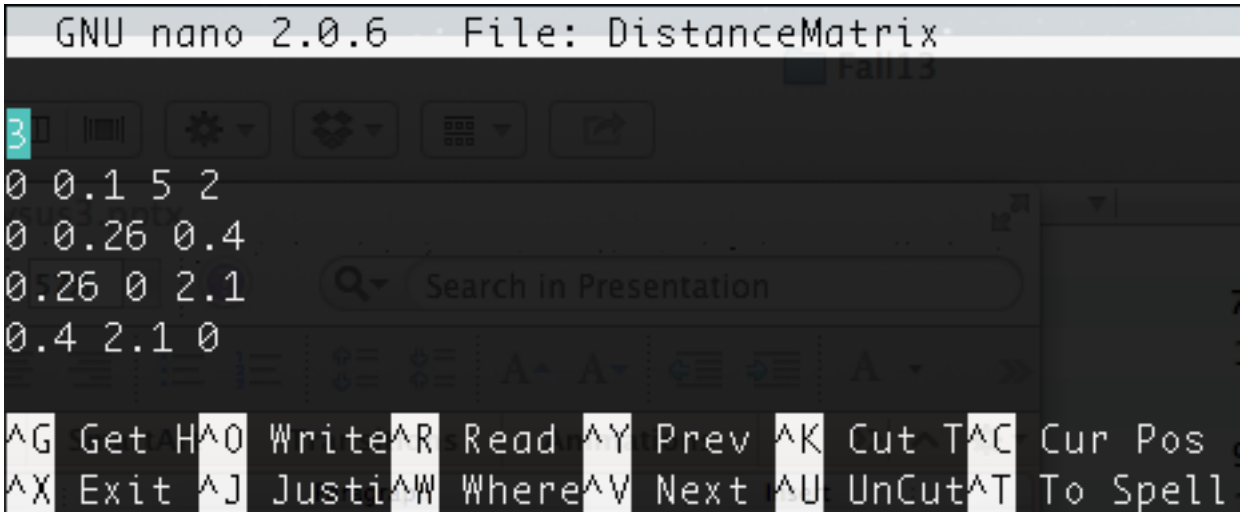
Pre-Compiled Executables

Downloads idarcy$ cd perseus_3_beta
perseus_3_beta idarcy$ g++ Pers.cpp -O3 -o perseus

Use any text editor (such as TextEdit or pico) to create input File. For example,

perseus_3_beta idarcy$ pico DistanceMatrix

Enter text



If you use pico, use control-X to exit and choose y to save (or control-O to save).

```
Downloads idarcy$ cd perseus_3_beta
perseus_3_beta idarcy$ g++ Pers.cpp -O3 -o perseus
perseus_3_beta idarcy$ pico DistanceMatrix
```

```
3
0 0.1 5 2
0 0.26 0.4
0.26 0 2.1
0.4 2.1 0
```

Number of data points.
I.e., size of matrix is 3x3

distance matrix

initial radius r = 0,
step size s = 0.1,
number of steps N = 5,
dimension cap C = 2

Increase radius
by 0.1 five times.

max dim of simplices

```
perseus_3_beta idarcy$ ./perseus rips DistanceMatrix

Read 2 point/radius pairs and birth times!
Writing Cell Complex From RIPS Complex
Done!Complex stored with 2 cells!
+++coreductions:  2 -> 2, fraction removed 0 at height 1
+++reductions:    2 -> 2, fraction removed 0 at height 2
Computing Persistence Intervals!
Linearly ordered 2 cells...
 Frame [0]: 1
 Frame [2]: 2

Done!!! Please consult [output*.txt] for results.
```

http://www.mrzv.org/software/dionysus/

Welcome to Dionysus' documentation!

Dionysus is a C++ library for computing persistent homology. It provides implementations of the following algorithms:

- Persistent homology computation [ELZ02] [ZC05]
- Vineyards [CEM06] (C++ only)
- Persistent cohomology computation (described in [dSVJ09])
- Zigzag persistent homology [CdSM09]
- *Examples* provide useful functionality in and of themselves:
  - *Alpha shape construction* in 2D and 3D
  - *Rips complex construction*
  - Cech complex construction (C++ only)
  - *Circle-valued parametrization*
  - *Piecewise-linear vineyards*

Contents

Special thanks to Dmitriy Morozov & Mikael Vejdemo-Johansson

Dependencies

http://www.mrzv.org/software/dionysus/get-build-install.html

Dionysus requires the following software:

CMake: for building (version $\geq$ 2.6)

Boost: C++ utilities (version $\geq$ 1.36; including Boost.Python used to create Python bindings)

Optional dependencies:

CGAL: for alpha shapes (version $\geq$ 3.4)

CVXOPT: for *circle-valued parametrization* using LSQR

PyQt4: for `viewer` module

The easiest way to download and install boost:

idarcy$ sudo port install boost

WARNING: Improper use of the sudo command could lead to data loss or the deletion of important system files. Please double-check yourtyping when using sudo. Type "man sudo" for more information.

To proceed, enter your password, or type Ctrl-C to abort.

Password:

Add path to your .bashrc:

use pico or other texteditor to add a pythonpath to your .bashrc file. If you are uncomfortable modifying your .bashrc file, you can create a copy first:

idarcy$ cp .bashrc .bashrcBACKUP

Add path to your .bashrc:

use pico or other texteditor to add the following line to the end of your .bashrc file:

my home is idarcy

```
export PYTHONPATH=$HOME/Dionysus/build/bindings/python
```

Recall I put Dionysus in idarcy:

idarcy$ hg clone http://hg.mrzv.org/Dionysus/

Add path to your .bashrc:

use pico or other texteditor to add the following line to the end of your .bashrc file:

my home is idarcy

```
export PYTHONPATH=$HOME/Downloads/
Dionysus/build/bindings/python
```

If I had put Dionysus in my Downloads folder:

```
Downloads idarcy$ hg clone http://hg.mrzv.org/
Dionysus/
```

Note this path is not yet active until you either open a new terminal or

directory_name idarcy$ cd
idarcy$ source .bashrc

cd takes me straight to my home directory no matter where I am.

To check if path is set correctly:

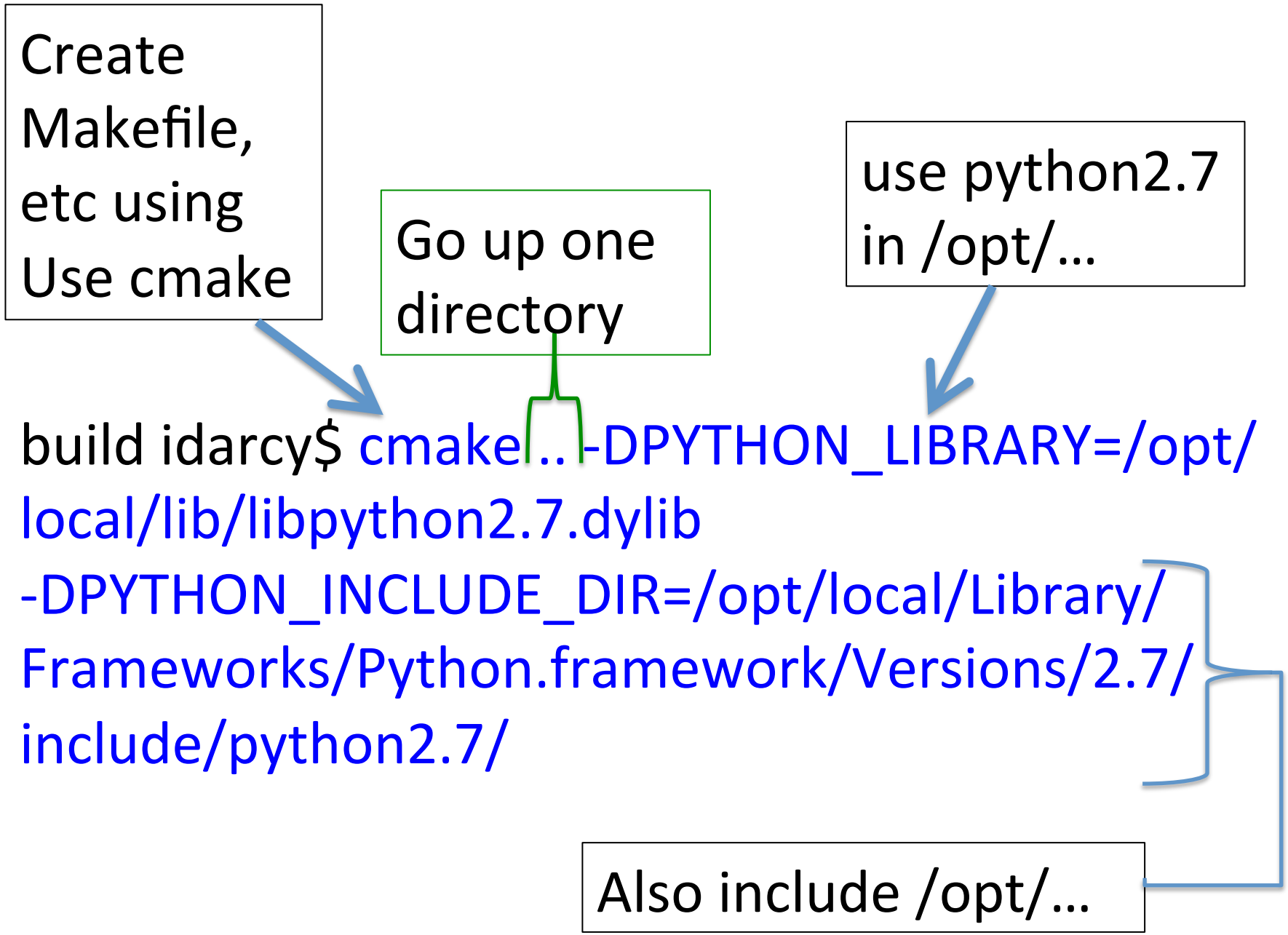
idarcy$ echo $PYTHONPATH
/Users/idarcy/Dionysus/build/bindings/python

idarcy$ cd Dionysus
Dionysus idarcy$ mkdir build
Dionysus idarcy$ cd build

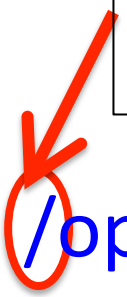
I had a python problem when I
build idarcy$ cmake ..
build idarcy$ make

I needed to specify which python to use:

build idarcy$ cmake .. -DPYTHON_LIBRARY=/opt/local/lib/libpython2.7.dylib -DPYTHON_INCLUDE_DIR=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/include/python2.7/


Create Makefile, etc using Use cmake
Go up one directory
use python2.7 in /opt/…
build idarcy$ cmake .. -DPYTHON_LIBRARY=/opt/local/lib/libpython2.7.dylib -DPYTHON_INCLUDE_DIR=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/include/python2.7/
Also include /opt/…

/ takes me straight to desired directory no matter where I am

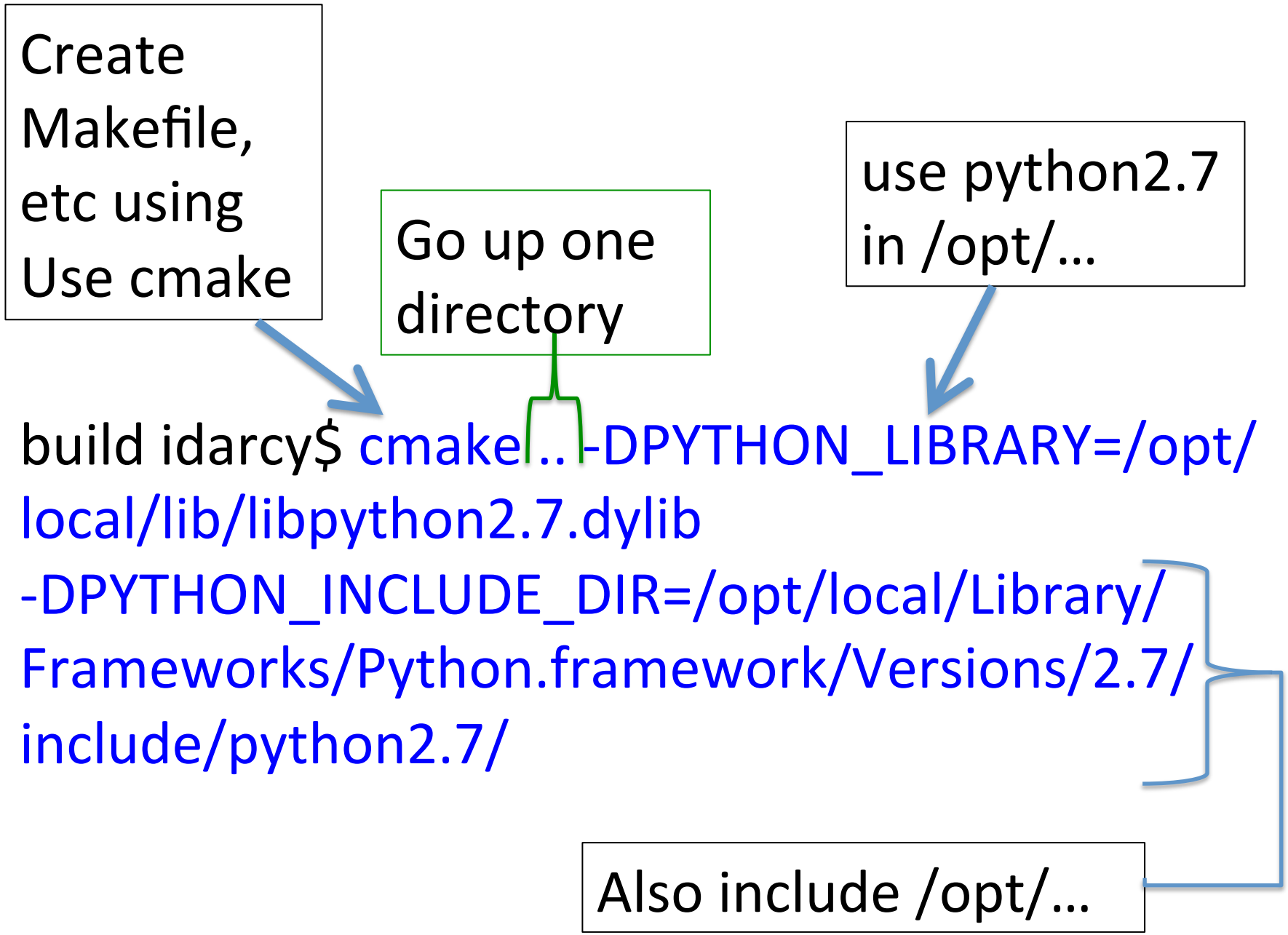


```
idarcy$ cd /opt/local/lib/
lib idarcy$ ls
lib idarcy$ ls *python*
libboost_python-mt.a      libboost_python-mt.dylib
    libpython2.7.dylib
```

* = wildcard

ls: lists all files

ls *python*: lists only files that contain python


Create Makefile, etc using Use cmake
Go up one directory
use python2.7 in /opt/…
build idarcy$ cmake .. -DPYTHON_LIBRARY=/opt/local/lib/libpython2.7.dylib -DPYTHON_INCLUDE_DIR=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/include/python2.7/
Also include /opt/…

```
idarcy$ port contents python27 | grep pyconfig
  /opt/local/Library/Frameworks/
Python.framework/Versions/2.7/include/
python2.7/pyconfig.h
```

build idarcy$ make

To run triangle example:

Method 1 from within build:

build idarcy$ cd examples/triangle/
triangle idarcy$ ./triangle

Method 2 from within Dionysus

../: Go up one directory

```
triangle idarcy$ pwd
/Users/idarcy/Dionysus/build/examples/triangle
triangle idarcy$ cd ../../../
Dionysus idarcy$ cd examples/triangle/
triangle idarcy$ python2.7 triangle.py
```

Recall that I am using python2.7. If you are using python, you can type python triangle.py

```
triangle idarcy$ cd
idarcy$ python2.7
Python 2.7.5 (default, Aug  1 2013, 01:01:17)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/
Apple/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> s = Simplex([0,1,2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Simplex' is not defined
>>> from dionysus import *
>>> s = Simplex([0,1,2])
```

cd takes me straight to my home directory no matter where I am.

Alternatively, can run python via command lines.

```
triangle idarcy$ python2.7 triangle.py
Complex: [<0>, <1>, <2>, <0, 1> 2.500000, <1, 2> 2.900000, <0, 2> 3.500000, <0, 1, 2>]
Vertex:  [<0>, <0, 1> 2.500000, <0, 1, 2>, <0, 2> 3.500000, <1>, <1, 2> 2.900000, <2>]
Data:    [<0>, <1>, <2>, <0, 1> 2.500000, <1, 2> 2.900000, <0, 2> 3.500000, <0, 1, 2>]
DataDim: [<0>, <1>, <2>, <0, 1> 2.500000, <1, 2> 2.900000, <0, 2> 3.500000, <0, 1, 2>]
Complex in the filtration order: <0>, <1>, <2>, <0, 1> 2.500000, <1, 2> 2.900000, <0, 2> 3.500000, <0, 1, 2>
Persistence initialized
Simplices paired
True True
<0> (1) - <0> (1)
Cycle (0):
True False
<1> (1) - <0, 1> 2.500000 (0)
Cycle (0):
True False
<2> (1) - <1, 2> 2.900000 (0)
Cycle (0):
False True
<0, 1> 2.500000 (0) - <1> (1)
Cycle (2): <1> + <0>
False True
<1, 2> 2.900000 (0) - <2> (1)
Cycle (2): <2> + <1>
True False
<0, 2> 3.500000 (1) - <0, 1, 2> (0)
Cycle (0):
False True
<0, 1, 2> (0) - <0, 2> 3.500000 (1)
Cycle (3): <0, 2> 3.500000 + <1, 2> 2.900000 + <0, 1> 2.500000
Number of unpaired simplices: 1
```

Method 2 from within Dionysus/examples/triangle

triangle.py

```
from dionysus import Simplex, Filtration,
StaticPersistence, \
                vertex_cmp, data_cmp, data_dim_cmp \

complex = [Simplex((0,),       0),              # A
        Simplex((1,),       1),              # B
        Simplex((2,),       2),              # C
        Simplex((0,1),      2.5),              # AB
        Simplex((1,2),      2.9),              # BC
        Simplex((0,2),      3.5),              # CA
        Simplex((0,1,2),    5)]              # ABC
```

triangle.py

```
print "Complex:", complex
```

```
triangle idarcy$ python2.7 triangle.py
Complex: [<0>, <1>, <2>, <0, 1> 2.500000, <1, 2>
2.900000, <0, 2> 3.500000, <0, 1, 2>]
```

```
print "Vertex: ", sorted(complex, vertex_cmp)
```

```
Vertex:  [<0>, <0, 1> 2.500000, <0, 1, 2>, <0, 2>
3.500000, <1>, <1, 2> 2.900000, <2>]
```

vertex_cmp(s1, s2)

Compares the two simplices with respect to the lexicographic order of their vertices.

```
print "Data:   ", sorted(complex, data_cmp)
Data:    [<0>, <1>, <2>, <0, 1> 2.500000, <1, 2>
2.900000, <0, 2> 3.500000, <0, 1, 2>]
```

data_cmp(s1, s2)

Compares the two simplices with respect to the data (real values) they store.

```
print "DataDim:", sorted(complex, data_dim_cmp)
DataDim: [<0>, <1>, <2>, <0, 1> 2.500000, <1, 2>
2.900000, <0, 2> 3.500000, <0, 1, 2>]
```

data_dim_cmp(s1, s2)

Compares the two simplices with respect to their dimension and within the same dimension with respect to their data

```
f = Filtration(complex, data_cmp)
print "Complex in the filtration order:", ', '.join((str(s)
for s in f))
```

```
Complex in the filtration order: <0>, <1>, <2>, <0, 1>
2.500000, <1, 2> 2.900000, <0, 2> 3.500000, <0, 1, 2>
```

```
p = StaticPersistence(f)
print "Persistence initialized"
```

```
Persistence initialized
```

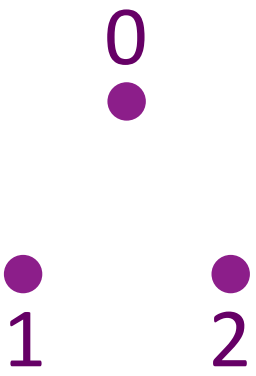
```
p.pair_simplices(True)
print "Simplices paired”
```

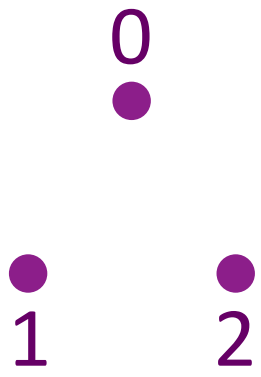
```
Simplices paired
```

```
pair_simplices(store_negative = False)
Pairs simplices using the [ELZ02] algorithm.
store_negative indicates whether to store the
negative simplices in the cycles.
```

```
smap = p.make_simplex_map(f)
for i in p:
    print i.sign(), i.pair().sign()
    print "%s (%d) - %s (%d)" % (smap[i], i.sign(),
                                 smap[i.pair()], i.pair().sign())
    print "Cycle (%d):" % len(i.cycle), "+"
                        .join((str(smap[ii]) for ii in i.cycle))
```

```
True True
<0> (1) - <0> (1)
Cycle (0):
```





True True

1 <0> (1) - <0> (1)

Cycle (0):

True False

2 <1> (1) - <0, 1> 2.500000 (0)

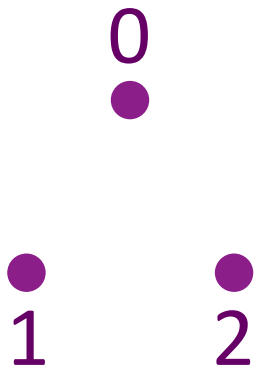
Cycle (0):

True False

3 <2> (1) - <1, 2> 2.900000 (0)

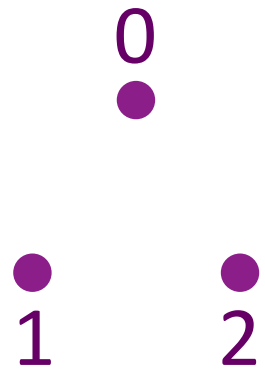
Cycle (0):

```
print "Cycle (%d):" % len(i.cycle), "+"
.join((str(smap[ii]) for ii in i.cycle))
```




```
False True
4 <0, 1> 2.500000 (0) - <1> (1)
Cycle (2): <1> + <0>
False True
5 <1, 2> 2.900000 (0) - <2> (1)
Cycle (2): <2> + <1>
True False
6 <0, 2> 3.500000 (1) - <0, 1, 2> (0)
Cycle (0):
```

```
print "Cycle (%d):" % len(i.cycle), "+"
.join((str(smap[ii]) for ii in i.cycle))
```



False True

7 <0, 1, 2> (0) - <0, 2> 3.500000 (1)

Cycle (3): <0, 2> 3.500000 + <1, 2> 2.900000
+ <0, 1> 2.500000

```
print "Number of unpaired simplices:", len([i for i in p if i.unpaired()])
```

```
Number of unpaired simplices: 1
```