Scientific Computing (Phys 2109/ Ast 3100H)

I. Scientfic Software Development

SciNet HPC Consortium

This Lecture

- Brief Intro Tour of Python for visualization and analysis
- Homework 1
- Introduction to Problem for weeks 2-4
- Valgrind, gdb
- Modular programming and Testing
- Start on Homework 2

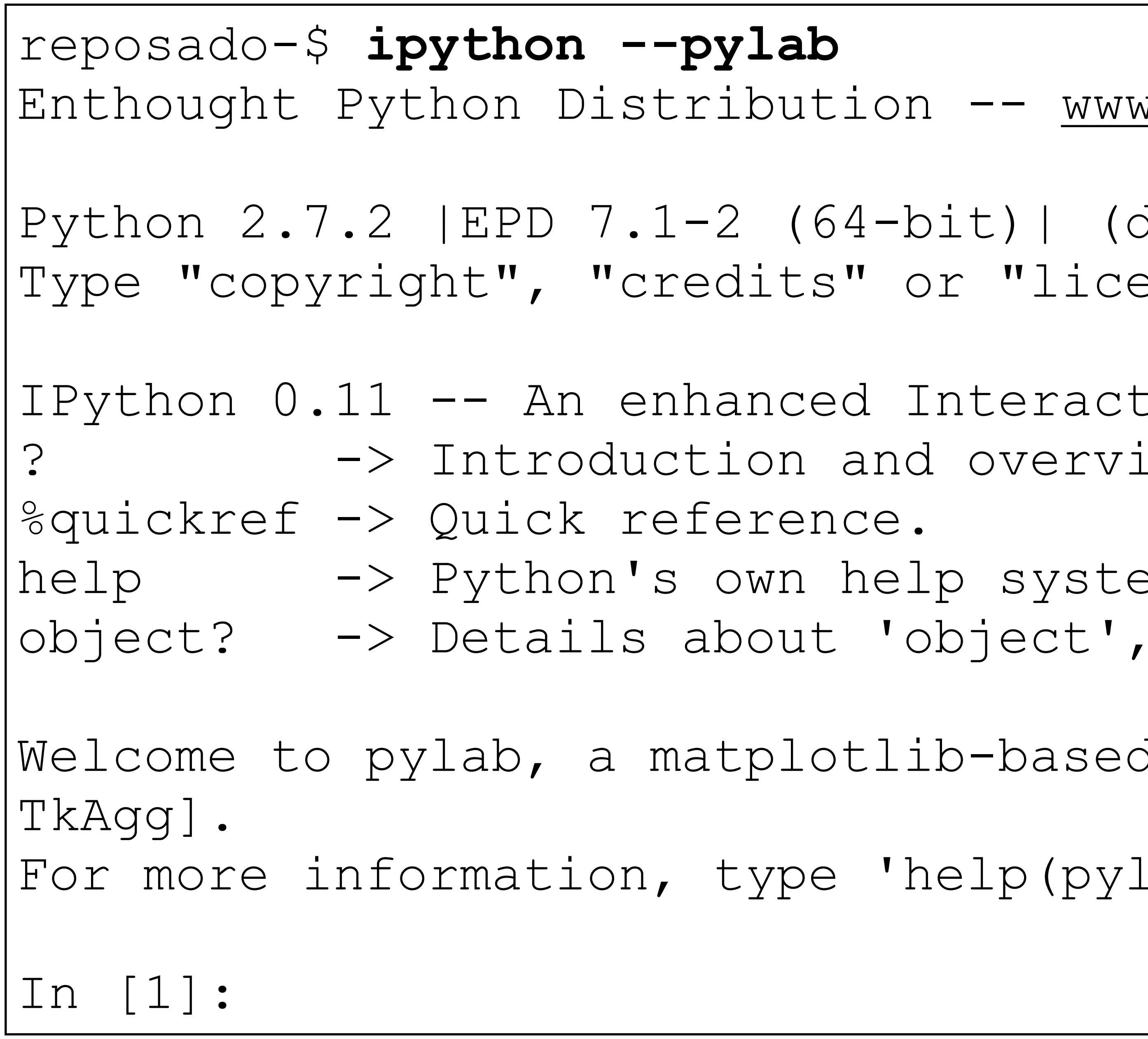
Python

- Flexible, mature (20yo) scripting-style programming language
- Ubiquitous
- Huge standard library, massive # of 3rd party packages
- Much slower than C/ Fortran or even IDL/Malab

http://www.python.org/

ipython

- For interactive use
- Automatically loads a lot of modules
  - If you write python scripts, have to do this on your own
- --pylab: lots of good math, plotting stuff.



```
reposado-$ ipython --pylab
Enthought Python Distribution -- www

Python 2.7.2 |EPD 7.1-2 (64-bit)| (d
Type "copyright", "credits" or "lice

IPython 0.11 -- An enhanced Interact
?          -> Introduction and overvi
%quickref -> Quick reference.
help       -> Python's own help syste
object?    -> Details about 'object',

Welcome to pylab, a matplotlib-based
TkAgg].
For more information, type 'help(pyl

In [1]:
```

Basic python

- Variables
- Like most scripting languages, don’t have to declare.
- Very handy for quick stuff, but has real drawbacks
- Math works the way you’d expect

```
In [1]: x = 2

In [2]: y = 3

In [3]: print x+y
5

In [4]: print x*y
6

In [5]: print y/x
1
```

Numpy, Arrays

- Python has lists [] but not "real" arrays
- Arrays are supplied by numpy, automatically included by pylab
- Numpy is the backbone of most scientific computing done in python.

```
In [6]: z = array([1., 2., 3., 4., 5.])

In [7]: print z
[ 1.  2.  3.  4.  5.]

In [8]: print x*z
[  2.   4.   6.   8.  10.]

In [9]: z2d = array([ [1.,2.,3.],
   ...:               [4.,5.,6.]] )

In [10]: print z2d
[[ 1.  2.  3.]
 [ 4.  5.  6.]]

In [11]: print y*z2d
[[  3.   6.   9.]
 [ 12.  15.  18.]]
```

Numpy, SciPy

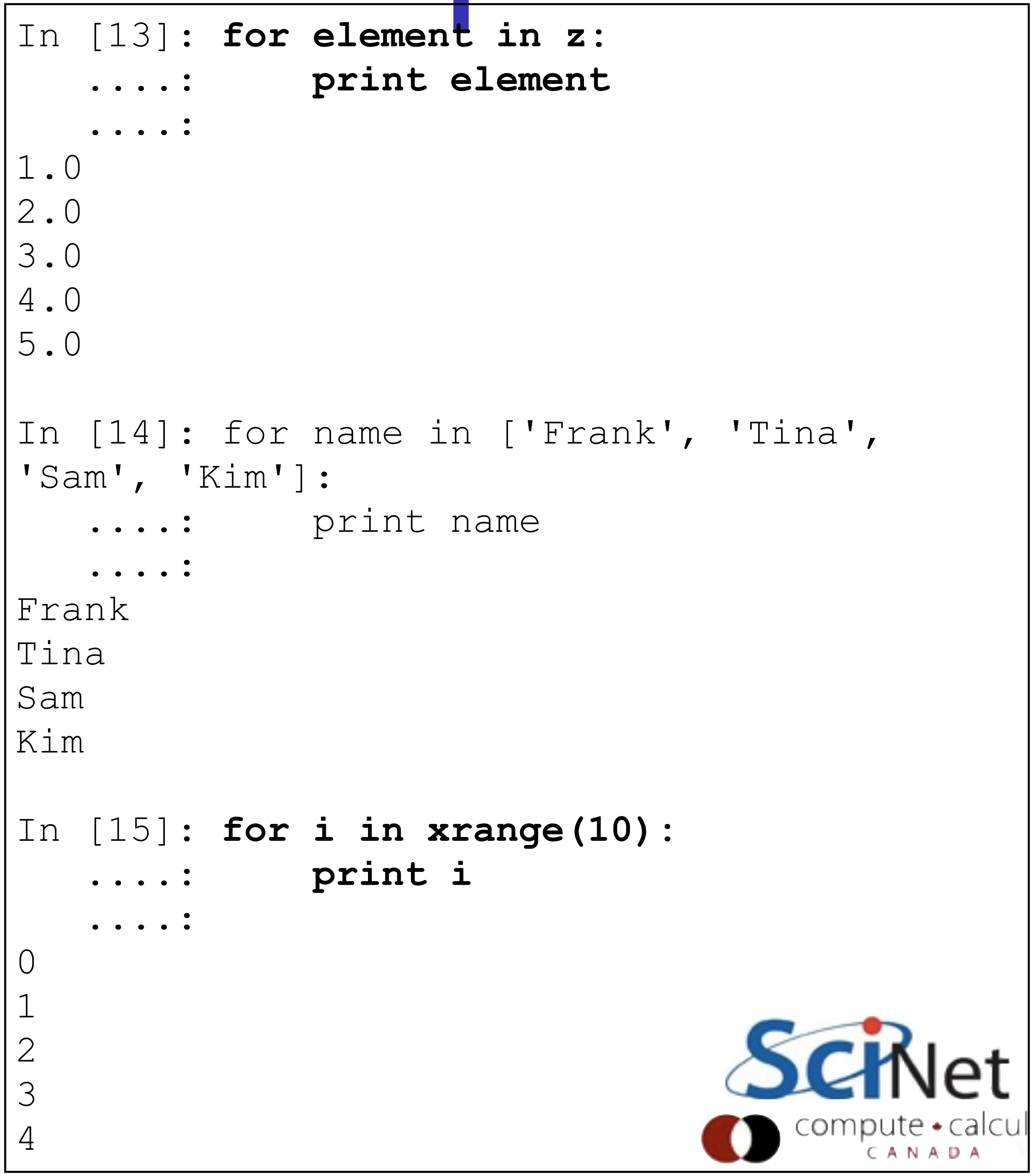
- Numpy provides basic N-dimensional array data structure, "fast" operations on that structure.
- Some low level math libraries
- SciPy has higher-level routines - linear algebra, fftpack, sparse matrix stuff, optimization packages, etc.

SciPy.org

http://www.scipy.org/SciPy

Python for loops

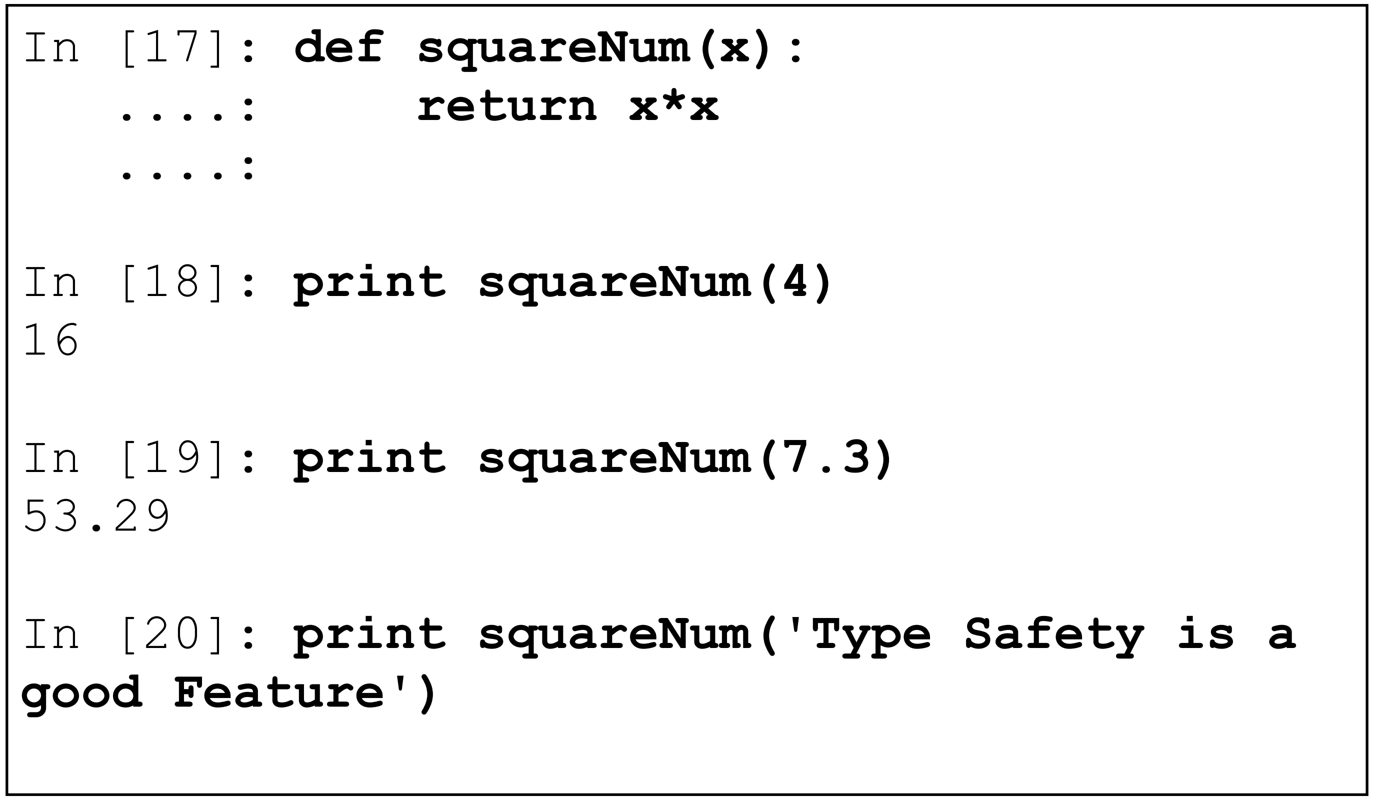
- For loops are more like "foreach"
- Each item in list
- If want a C-like for loop, use xrange (generates list 0..N-1)
- Note indentation: indentation is important in python!
- (what happens with for element in z2d?)



```
In [13]: for element in z:
   ....:     print element
   ....:
1.0
2.0
3.0
4.0
5.0

In [14]: for name in ['Frank', 'Tina',
'Sam', 'Kim']:
   ....:     print name
   ....:
Frank
Tina
Sam
Kim

In [15]: for i in xrange(10):
   ....:     print i
   ....:
0
1
2
3
4
```

Python Functions

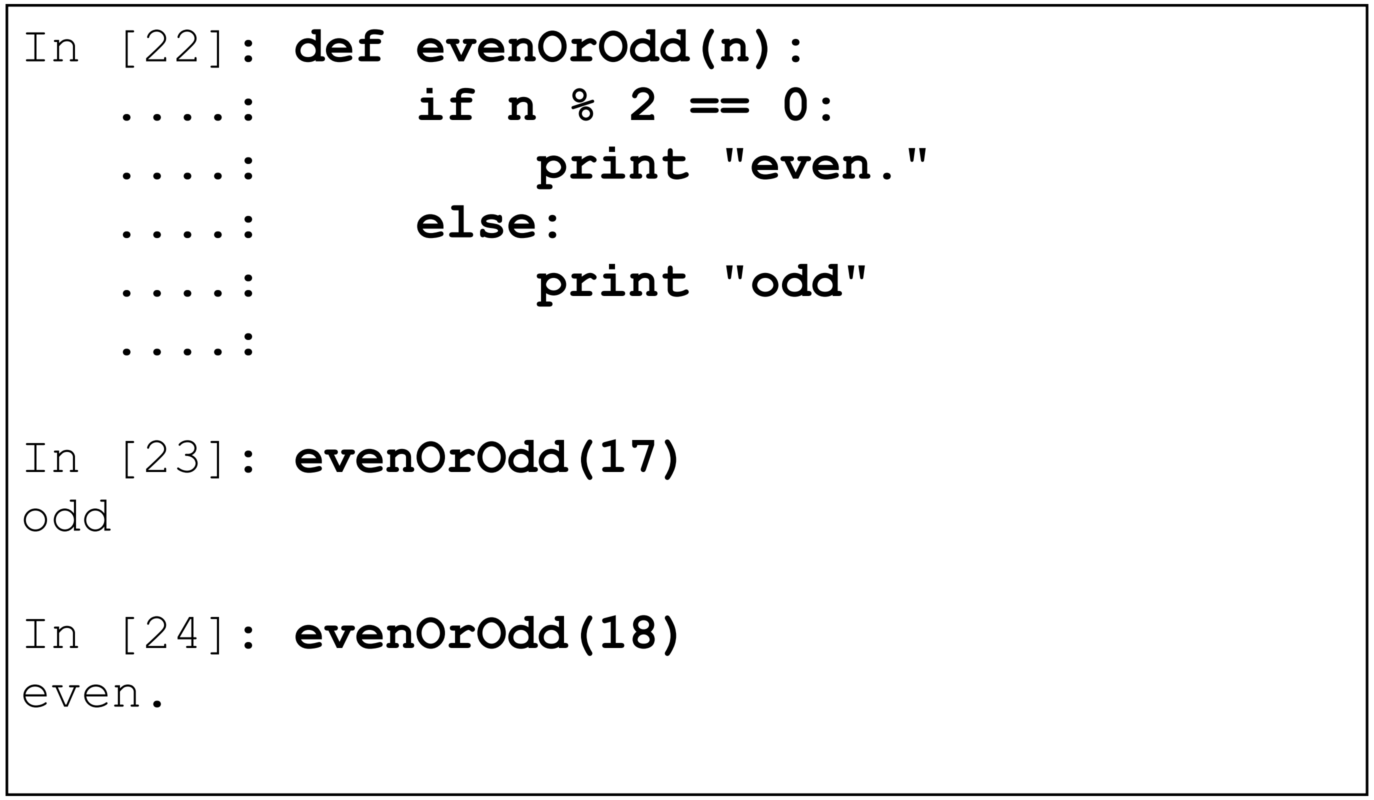
- Can also define functions
- ‘def’ keyword



```
In [17]: def squareNum(x):
   ....:     return x*x
   ....:

In [18]: print squareNum(4)
16

In [19]: print squareNum(7.3)
53.29

In [20]: print squareNum('Type Safety is a 
good Feature')
```

If/else

- Control flow
- Same :, same punctuation significance
- functions needn't return a value.



```
In [22]: def evenOrOdd(n):
   ....:     if n % 2 == 0:
   ....:         print "even."
   ....:     else:
   ....:         print "odd"
   ....:

In [23]: evenOrOdd(17)
odd

In [24]: evenOrOdd(18)
even.
```

Writing python files

- Can write functions in a file, import them in ipython
- specify them with filename.functionname
- Code not in functions will be run at import time.

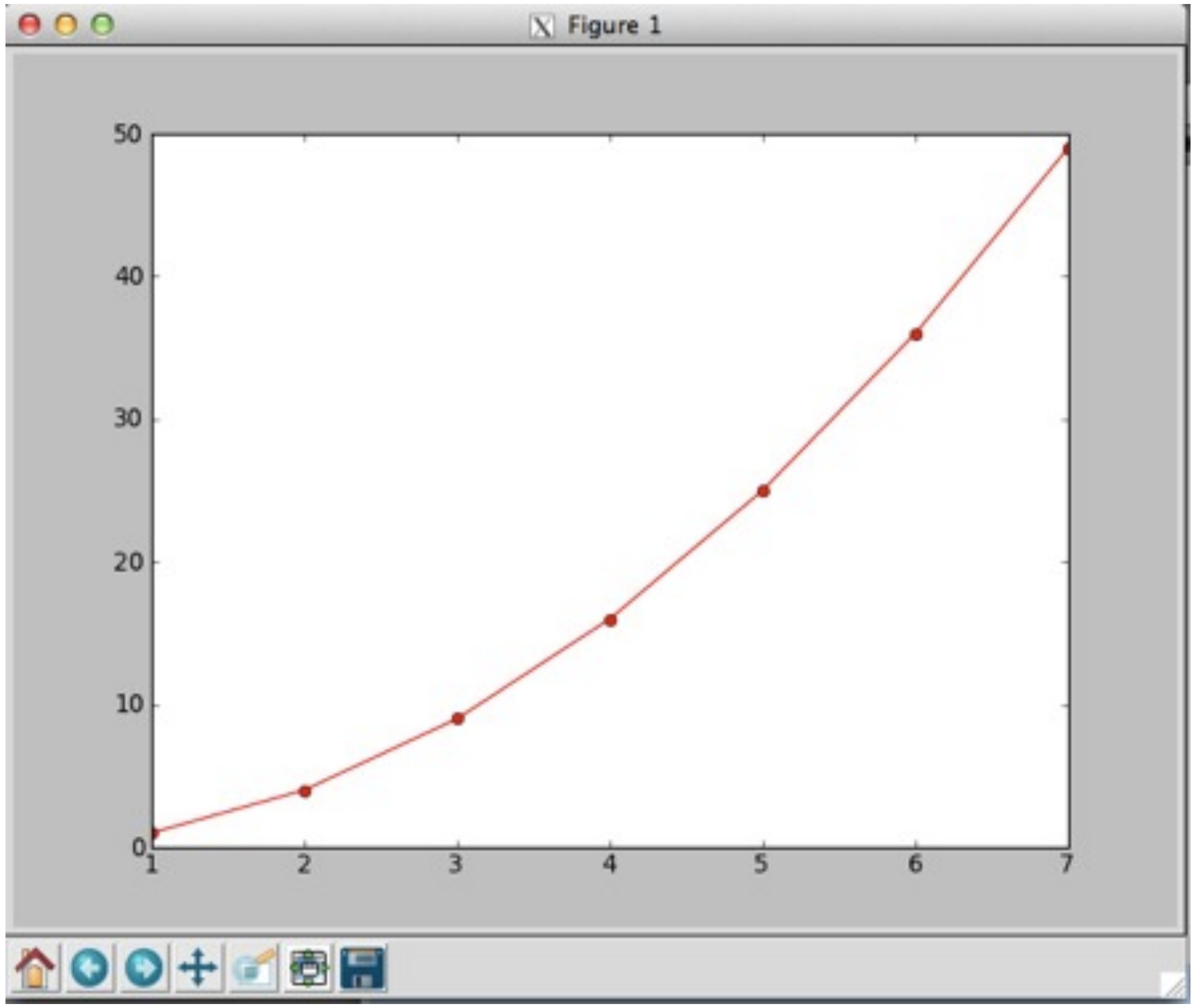
```
$ cat myRoutines.py
def myFunction(x, y):
    '''This returns square of sum of args'''
    return x*x+y*y
```

```
In [26]: import myRoutines

In [27]: help("myFunction")
...
FUNCTIONS
    myFunction(x, y)
        This returns square of sum of args

In [28]: a = myRoutines.myFunction(1, 2)

In [29]: print a
5
```

Basic Plotting with Matplotlib

- matplotlib.sourceforge.net/
- gallery of examples with source code
- matlab like

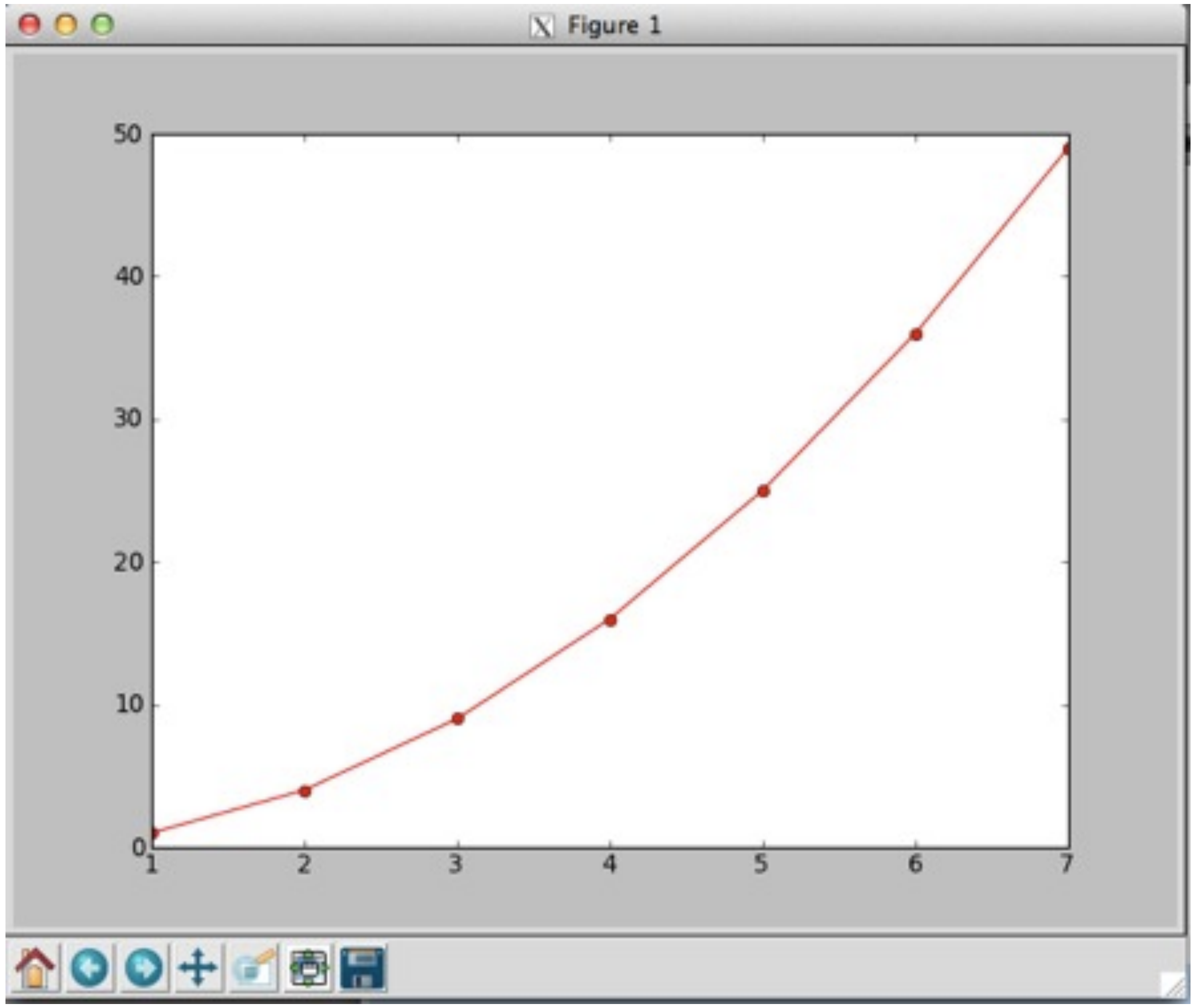
```
In [29]: x = array([1.,2.,3.,4.,5.,6.,7.])

In [30]: y = x*x

In [31]: plot(x,y)
Out[31]: [<matplotlib.lines.Line2D at ...

In [32]: clf()

In [33]: plot(x,y,'ro-')
Out[33]: [<matplotlib.lines.Line2D ...]
```



Basic Plotting with Matplotlib

- matplotlib.sourceforge.net/
- gallery of examples with source code
- matlab like

```
In [29]: x = array([1.,2.,3.,4.,5.,6.,7.])

In [30]: y = x*x

In [31]: plot(x,y)
Out[31]: [<matplotlib.lines.Line2D at ...

In [32]: clf()

In [33]: plot(x,y,'ro-')
Out[33]: [<matplotlib.lines.Line2D ...]
```

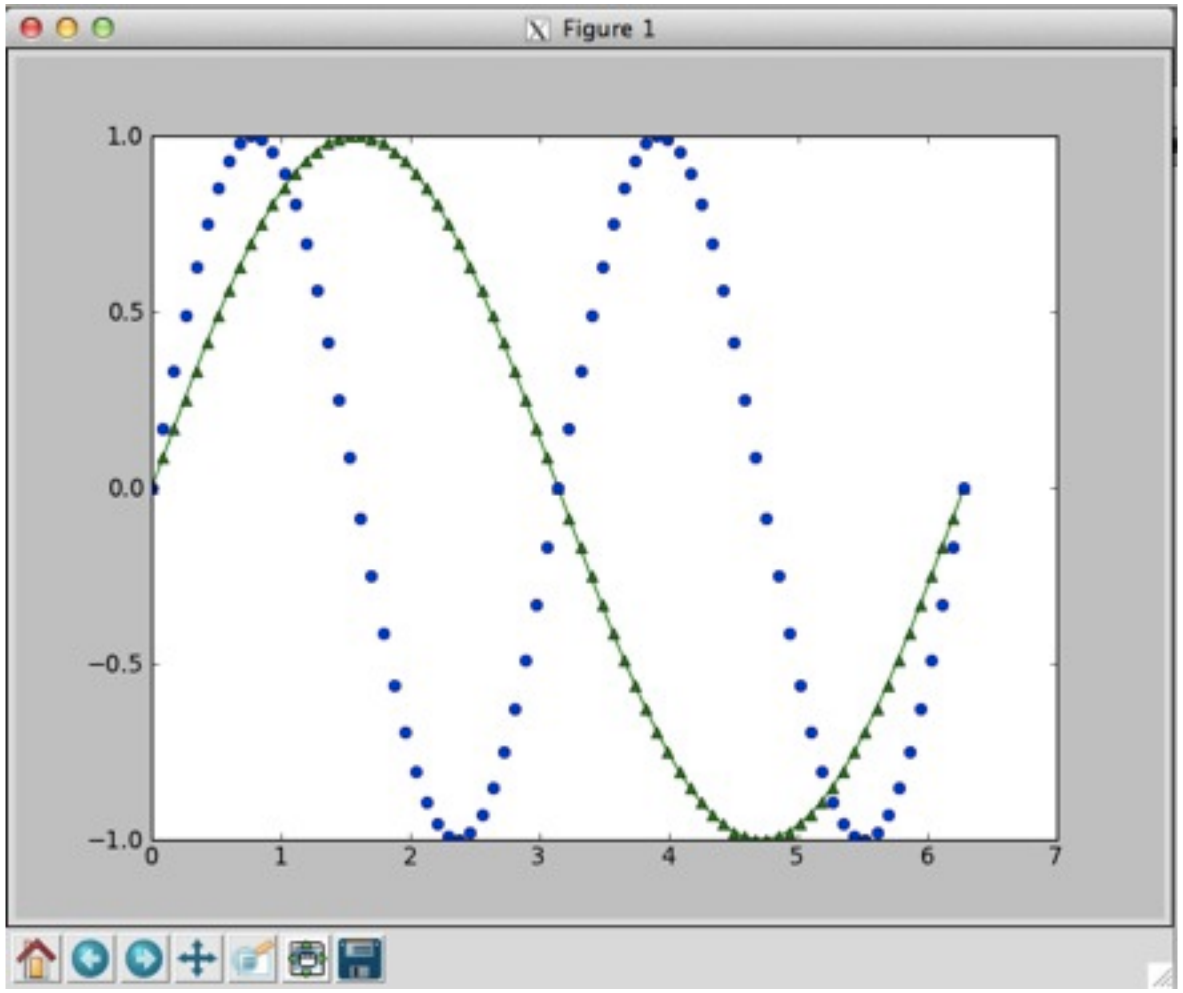


SciNet
compute • calcul
CANADA

Basic Plotting with Matplotlib

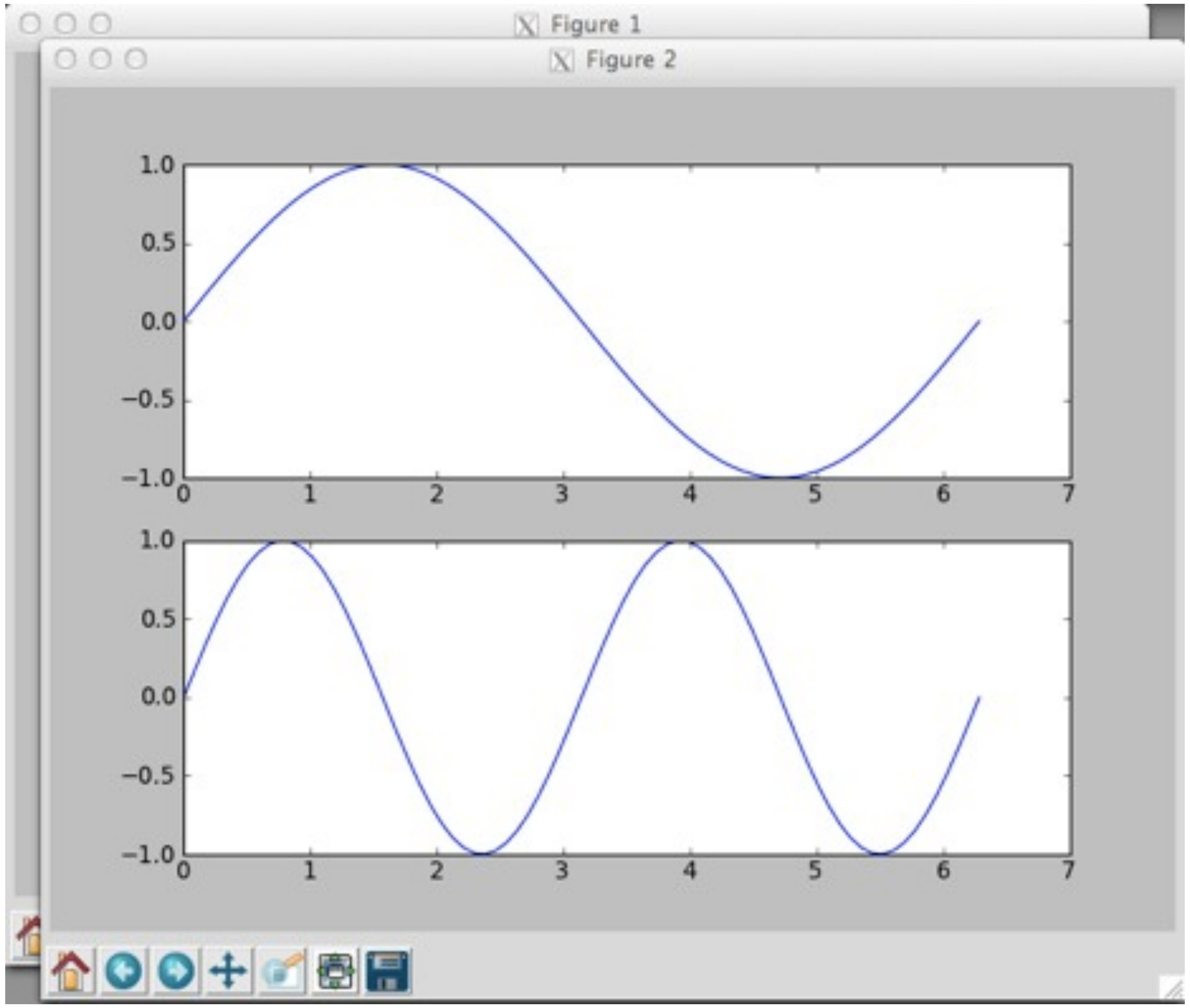
- linspace(start, end, npts)
- pi, e defined
- By default, overplot

```
In [34]: x = linspace(0,2*pi,75)

In [35]: y = sin(x)

In [36]: z = sin(2*x)

In [37]: plot(x, y, 'g^-')
Out[37]: [<matplotlib.lines.Line2D at
0x334d550>]

In [38]: plot(x, z, 'bo')
Out[38]: [<matplotlib.lines.Line2D at
0x3351b50>]
```



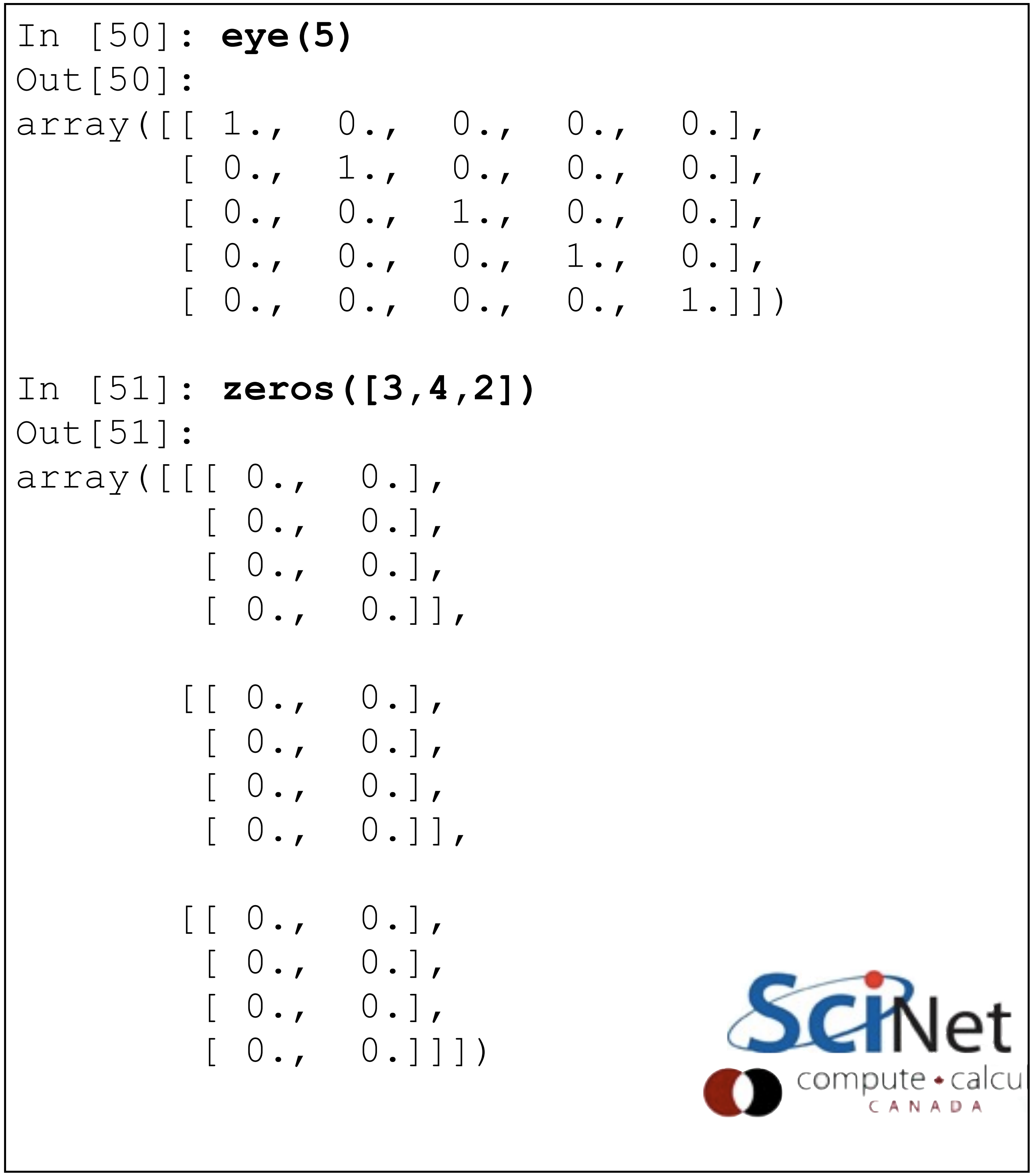
Multiple Figure Plotting

```
In [43]: figure()
Out[43]: <matplotlib.figure.Figure ...

In [44]: subplot(2,1,1)
Out[44]: <matplotlib.axes.AxesSubplot ...

In [45]: plot(x,y)
Out[45]: [<matplotlib.lines.Line2D ...

In [46]: subplot(2,1,2)
Out[46]: <matplotlib.axes.AxesSubplot ...

In [47]: plot(x,z)
Out[47]: [<matplotlib.lines.Line2D ...

In [48]: close()
```



Multi-d arrays

- By hand, as before
- Some special arrays: identity matrix of size nxn, or arbitrary shape array of zeros



```
In [50]: eye(5)
Out[50]:
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])

In [51]: zeros([3,4,2])
Out[51]:
array([[[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]],

       [[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]],

       [[ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.],
        [ 0.,  0.]]])
```

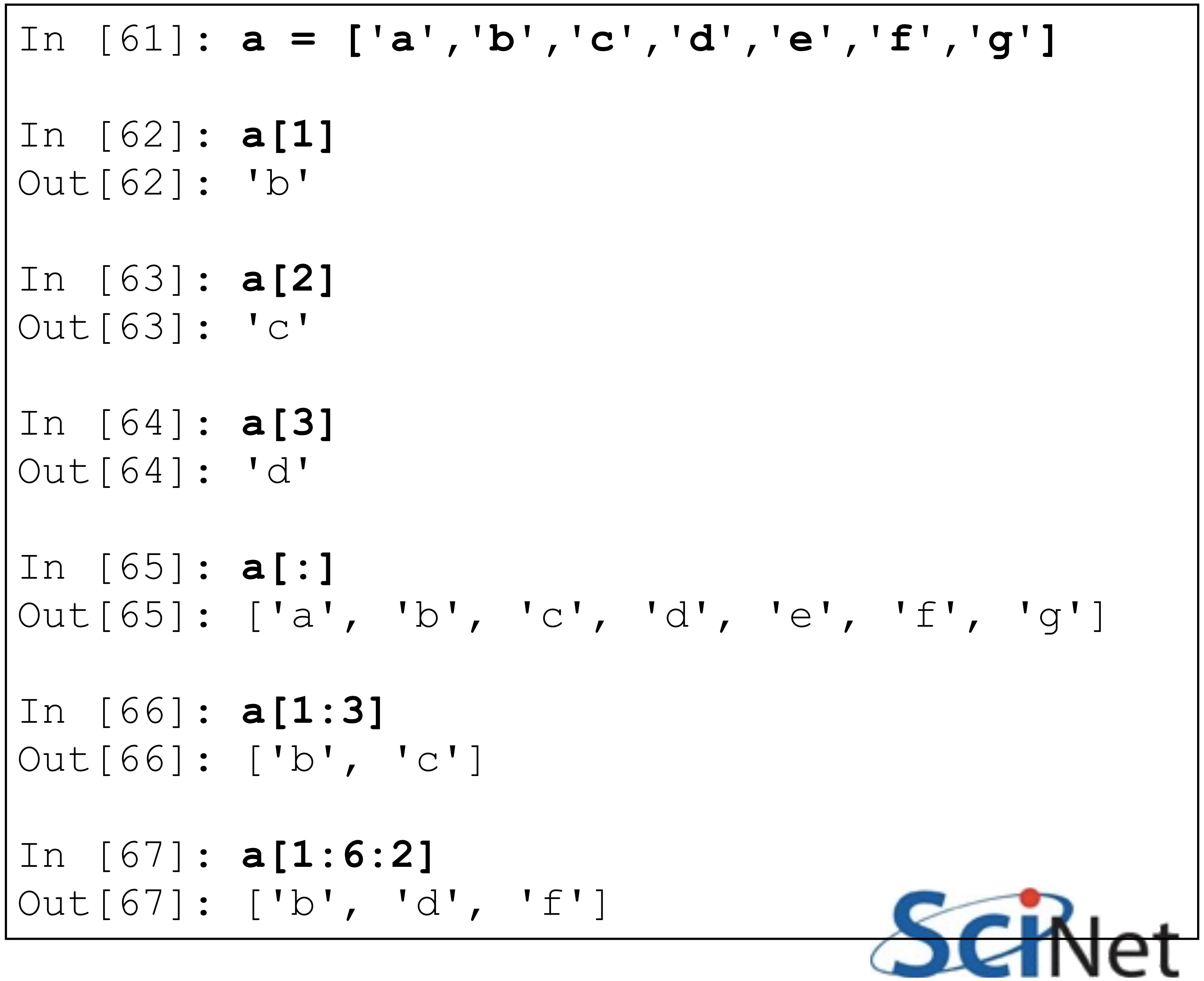
Multi-d arrays

- Python lists, numpy arrays, are zero-based
- Can select out particular rows, columns

```
In [55]: z = zeros([4,3])

In [56]: z[2,1] = 1.

In [57]: print z
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  0.]]

In [58]: print z[:,1]
[ 0.  0.  1.  0.]
```

Python Slicing

- Like Fortran, Matlab with one important difference
- ':' selects entire range in that dimension
- start:end selects from start to *before* end
- start:end:stride



```
In [61]: a = ['a','b','c','d','e','f','g']

In [62]: a[1]
Out[62]: 'b'

In [63]: a[2]
Out[63]: 'c'

In [64]: a[3]
Out[64]: 'd'

In [65]: a[:]
Out[65]: ['a', 'b', 'c', 'd', 'e', 'f', 'g']

In [66]: a[1:3]
Out[66]: ['b', 'c']

In [67]: a[1:6:2]
Out[67]: ['b', 'd', 'f']
```

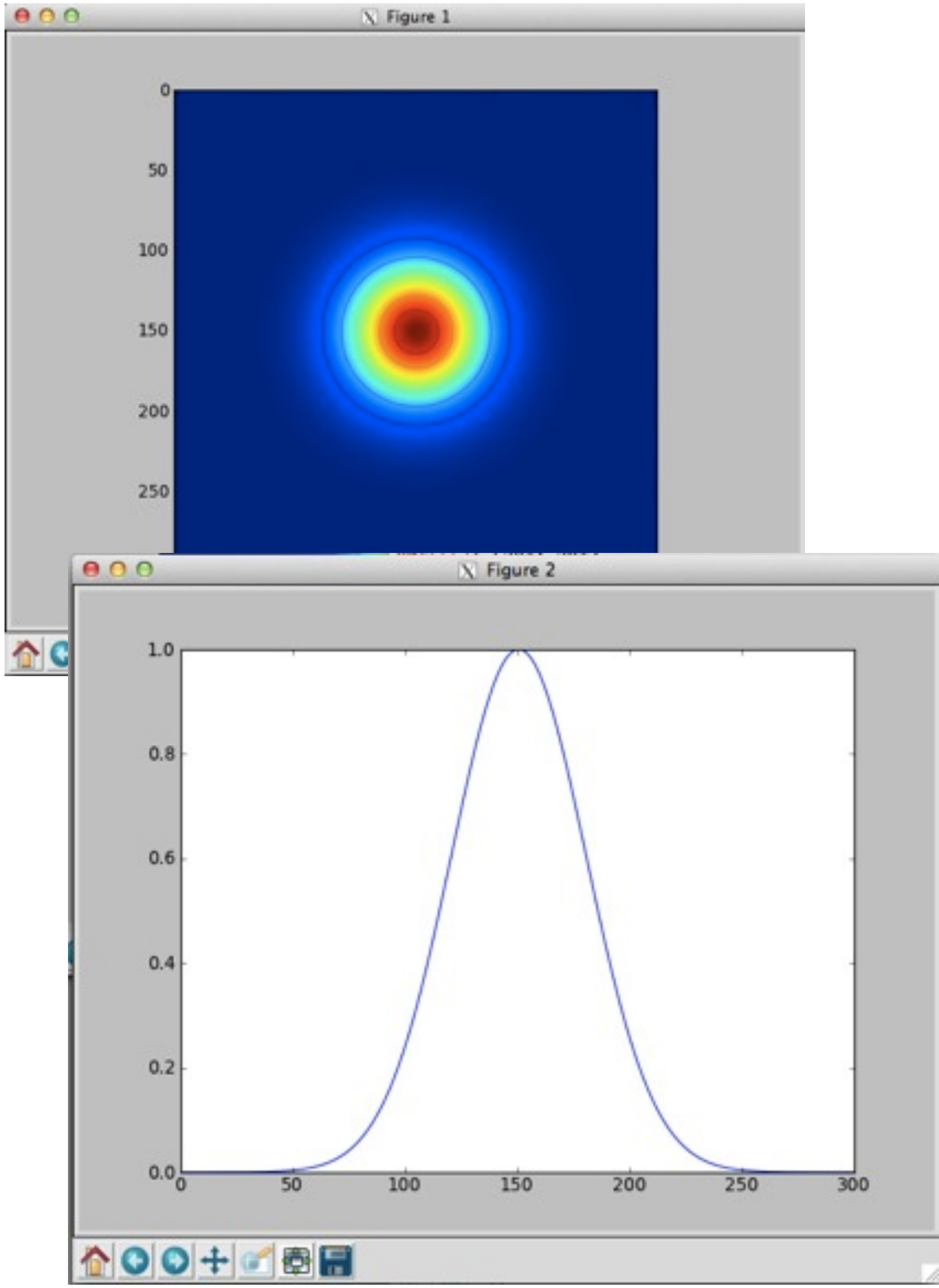
2d plotting

- First, let's load some 2d data
- Import your data from HW1
- (loaddata is another useful from-text-file routine)
- mgrid - generate x,y coordinates for 2d grid

```
In [76]: data = gen<TAB>
generic      genfromtxt

In [76]: data = genfromtxt('data.txt')

In [77]: shape(data)
Out[77]: (301, 301)

In [78]: x, y = mgrid[0:301,0:301]

In [79]: x.max()
Out[79]: 300

In [80]: x = x - 150.

In [81]: y = y - 150.

In [82]: r2 = x*x+y*y

In [83]: gauss = exp(-r2/(2*30.*30.))
```

2d plotting

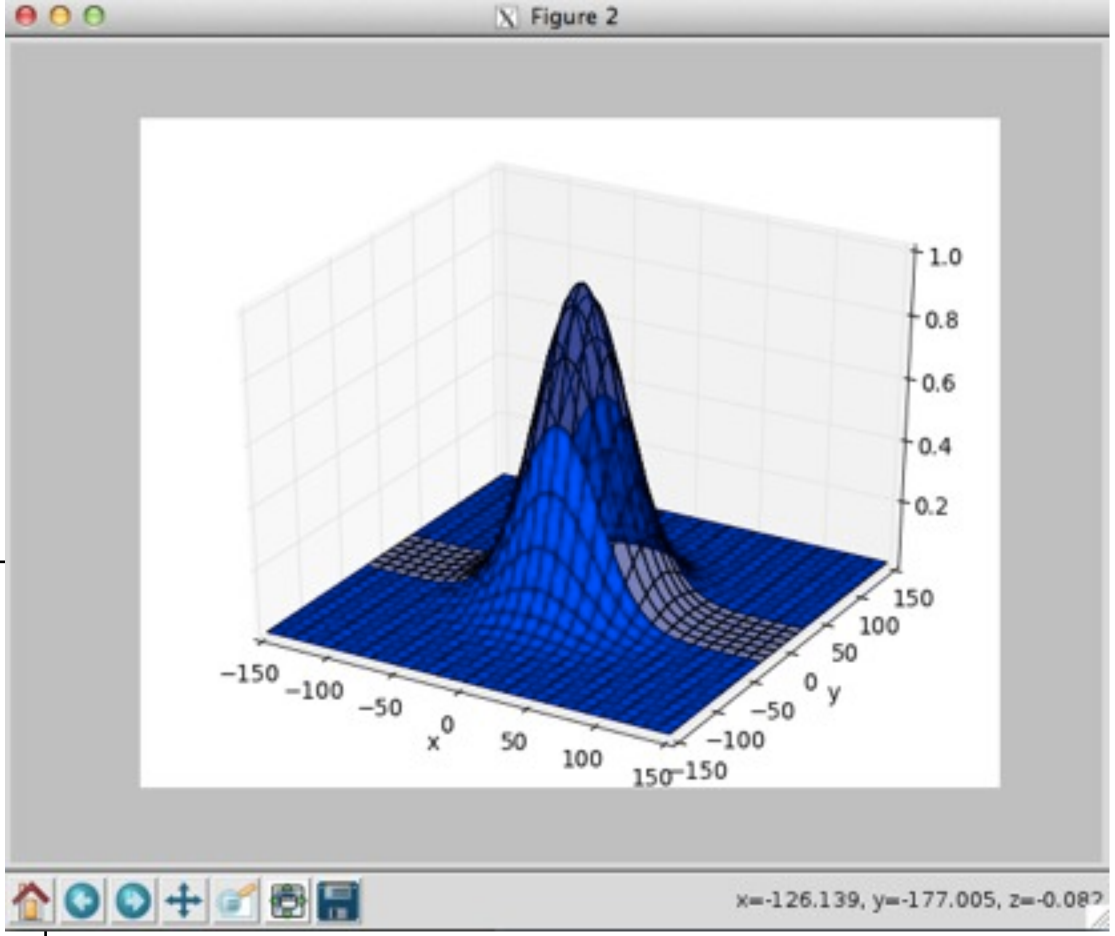
```
In [84]: clf()

In [85]: contour(data)
Out[85]: <matplotlib.contour.QuadContourSet
instance at 0x3751050>

In [86]: imshow(data)
Out[86]: <matplotlib.image.AxesImage at
0x3757f90>

In [87]: figure()
plot(Out[87]: <matplotlib.figure.Figure at
0x3757b50>

In [88]: plot(data[151,:])
Out[88]: [<matplotlib.lines.Line2D at
0x3f77ad0>]
```



3d plotting

- Lots of very powerful things possible with matplotlib
- But once you leave the simple things, starts getting cryptic.

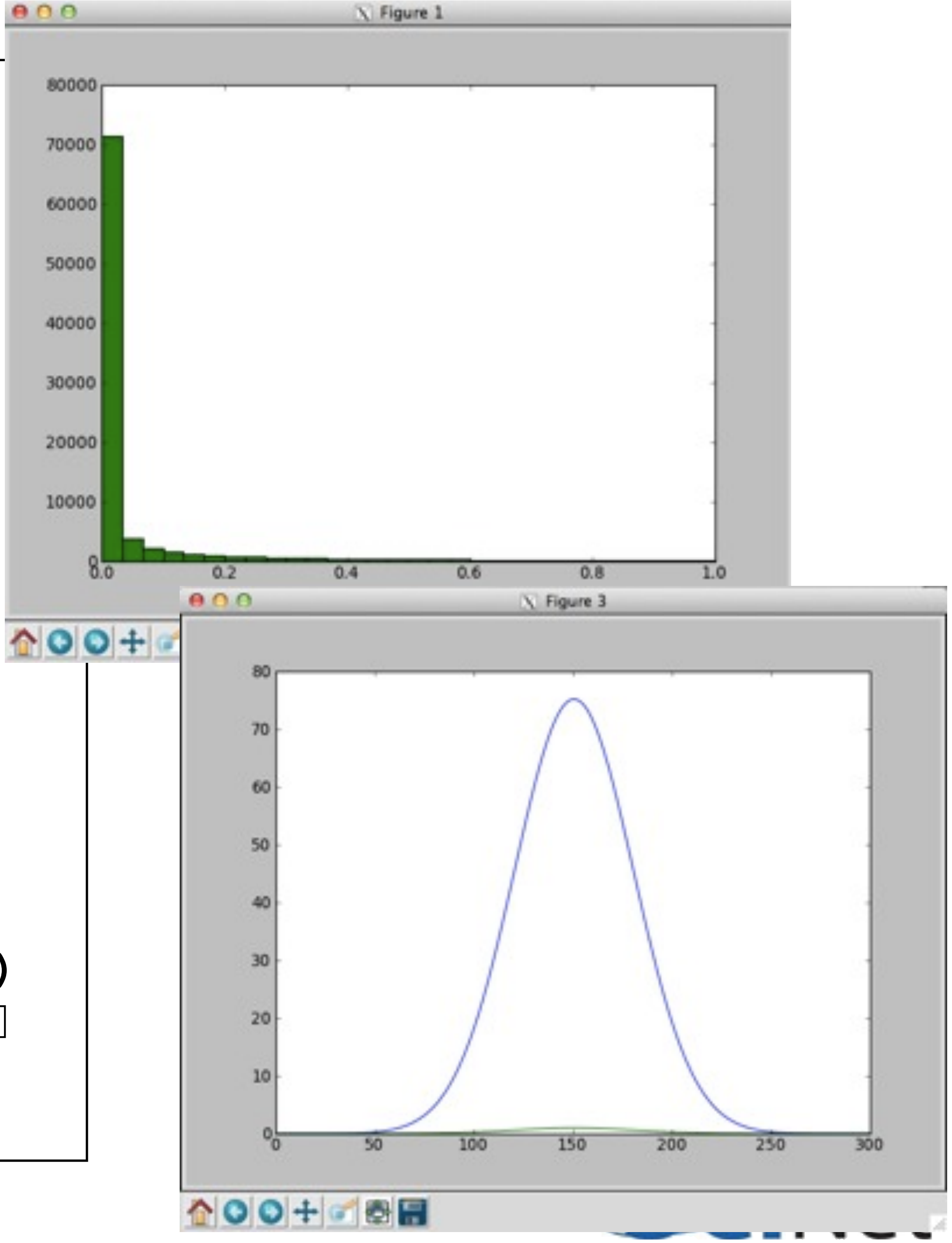
```
In [99]: from mpl_toolkits.mplot3d import Axes3D

In [100]: fig = figure()

In [101]: ax = fig.gca(projection='3d')

In [102]: ax.plot_surface(x,y,gauss)
xlOut[102]:
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at
0x3be3410>

In [103]: xlabel('x'); ylabel('y')
Out[103]: <matplotlib.text.Text at 0x4ea8590>
```



x=-126.139, y=-177.005, z=-0.082

Misc. Analysis

```
In [111]: hist(data.flatten(),30)
Out[111]:
(array([71365,   3904,   2316,   1620,   1236,   ...

In [115]: size(data)
Out[115]: 90601

In [116]: size(where(data > 0.2))
Out[116]: 18224

In [117]: size(where(data > 0.5))
Out[117]: 7816

In [118]: figure()
plot(Out[127]: <matplotlib.figure.Figure at
0x2bbb58d0>

In [128]: plot(sum(data,axis=1)); plot(data[151,:])
Out[128]: [<matplotlib.lines.Line2D at 0x2bbba950>]

In [129]: quit()
```



Homework 1

- We've reviewed all the homeworks; well done!
- Will have proper marks next week.
- Make, git quite well done.
- Biggest problem: .c / .h

Interface vs. Implementation

- The implementation - actual code - goes in the .c file.
- The interface - what the calling code needs to know about - goes in the .h file.
- This distinction is **crucial** for writing modular code.

What does main.c need to know at compile time?

```
$ cat outputarray.h

void output2dbin(char *filename, double **data, const int nrows, const int ncols);
void output2dascii(char *filename, double **data, const int nrows, const int ncols);
```

```
$ cat main.c

#include "outputarray.h"

int main(int argc, char **argv) {
  // ...
        tick(&clock);
        output2dascii("data.txt", data, nrows, ncols);
        asciitime = tock(&clock);

        tick(&clock);
        output2dbin("data.bin", data, nrows, ncols);
        bintime = tock(&clock);
  //
}
```

Interface vs. Implementation

- When main.c is being compiled to a .o file, needs to know that there exists out there somewhere a function of the form
  ```
  void output2dascii(char *, double **, const int,
  const int);
  ```
- Does not need to know implementation details (source of routine)
- **Neither does programmer of main.c**

Compiling vs. Linking

- main.o can’t be executed - it’s missing the routines for output2dascii() (and printf, and exp, and..)
- At link time, .o’s (or libraries) must be linked in to the executable that satisfy all those routines that the code needs.
- If you leave out one of the needed .o’s, fatal error - ‘symbol not found’

What goes in interface?

- At the very least, the function prototypes (so compiler can make sure it’s valid function, arguments)
- There may also be constants that calling function and routine need to agree on (eg, error codes) or definitions of data structures.

```
$ cat outputarray.h

void output2dbin(char *filename, d
void output2dascii(char *filename,
```

What goes in interface?

- Not necessarily every function prototype (or constant, or..)
- Usually, one .c/.h file per unit of functionality - often more than one routine.
- Internal routines do **not** get publicly exposed

```
$ cat outputarray.h

void output2dbin(char *filename, d
void output2dascii(char *filename,
```

Why does it matter?

- Scientific software can be large, complex, subtle.
- If each section uses internal details from each other section, have to understand the whole code at once to do everything
- Interactions grow as (Lines of code)^2.
- This is why global variables are bad
- **Have** to enforce boundaries between sections of code - self-contained modules of functionality.
- Makes testing easier

More work up front

- Think about what you want the pieces of functionality to be.
- How are you going to use these routines?
- Think about everything you might want to use these routines for, *then* design interface.
- May change a bit in early stages, but if it changes a lot you should rethink things - you're not using the functionality the way you thought.
- Like documentation, etc.. - more work upfront, much more productivity in long run.

Module design

- Keep purpose of module clear
- As simple as possible (for your own sanity)
- As general as makes sense

HW1 - Makefiles

- Makefiles were good, but don't forget header file dependancies (depend on interface to code, not implementation).
- If interface changes, code calling it will have to be recompiled
- gcc -MM can help:

```
$ gcc -MM main.c
main.o: main.c array2d.h gaussian.h outputarray.h
```

HW1 - Text vs Binary

- In HW1 sample soln on wiki, include two file outputs - in Text format describe, and in binary
- Also have timing of output:

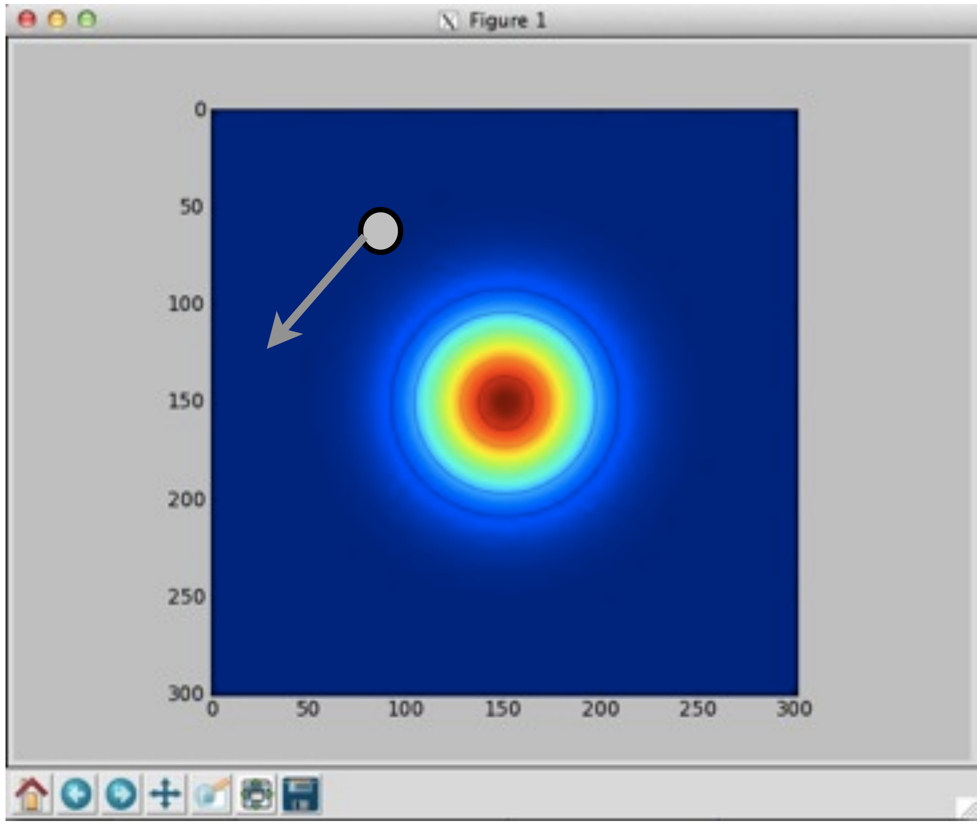
```
reposado-$ ./main 300 30
Text time = 0.073281, Binary time = 0.024263

reposado-$ ./main 3000 30
Text time = 6.368578, Binary time = 0.956578

reposado-$ ls -l data.*
-rw-r--r-- 1 ljdursi scinet  720008 Nov 10 21:50 data.bin
-rw-r--r-- 1 ljdursi scinet 1260300 Nov 10 21:50 data.txt
```

Text vs Binary

- Text ok for what we're doing - small
- Basically, ok for anything you might actually plausibly read.
- Not going to read it (15GB of data?) Binary.
- **Faster**, smaller.
- Accuracy!
- Number of good formats

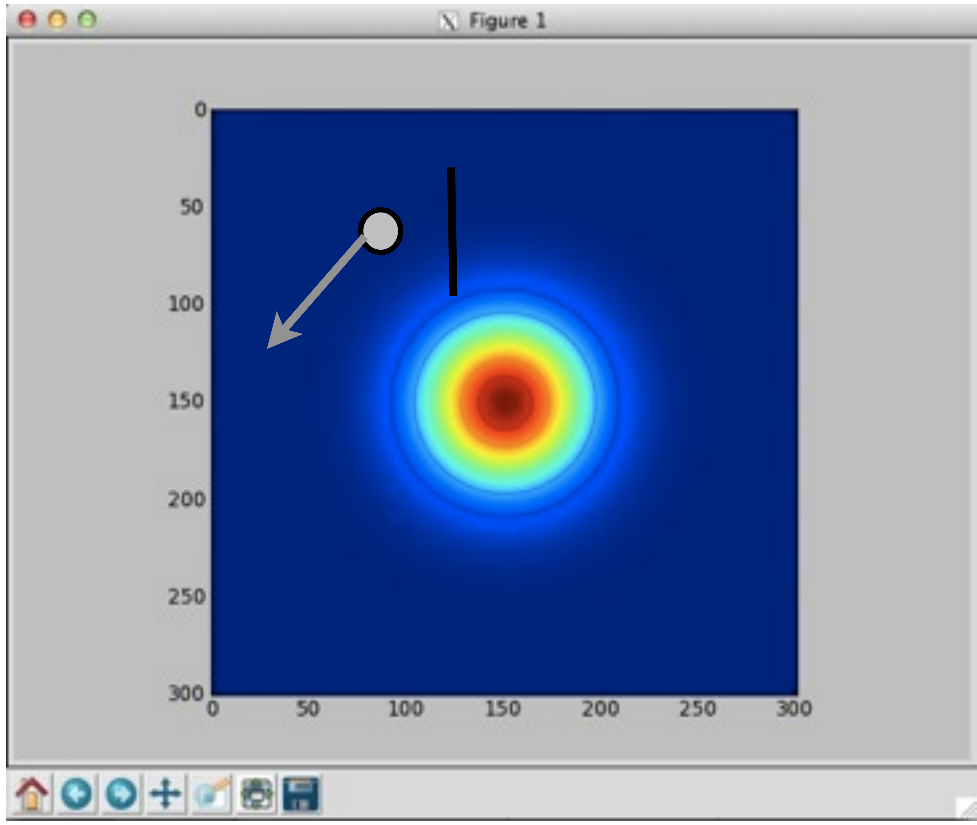
Course Project

- Will be working on for next 3 weeks
- Charged tracer particle moving in a diffusive environment
- Colloidal transport in fluid medium
- Couple kinds of physics, couple kinds of data structures (grid, particle)



Course Project

- Get source code:
- https://wiki.scinethpc.ca/wiki/images/f/fb/Diffuse2.c
- Setup: Supervisor has this old code for diffusive background, “works fine”, wants you to add tracer particle to it.
- Uses library you don’t have for ploting - ifdef’ed out for now.



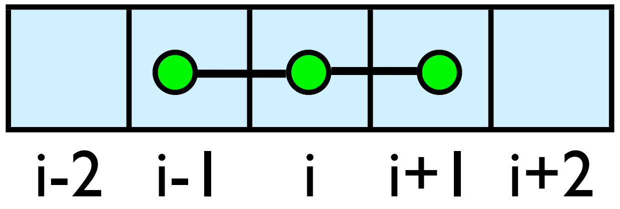
Discretizing Derivatives

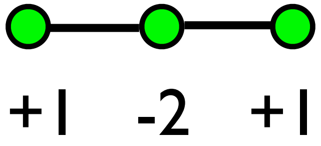
$$\left.\frac{d^2Q}{dx^2}\right|_i \approx \frac{Q_{i+1} - 2Q_i + Q_{i-1}}{\Delta x^2}$$

- Done by finite differencing the discretized values

- Implicitly or explicitly involves interpolating data and taking derivative of the interpolant

- More accuracy - larger ‘stencils’





Discretizing Derivatives

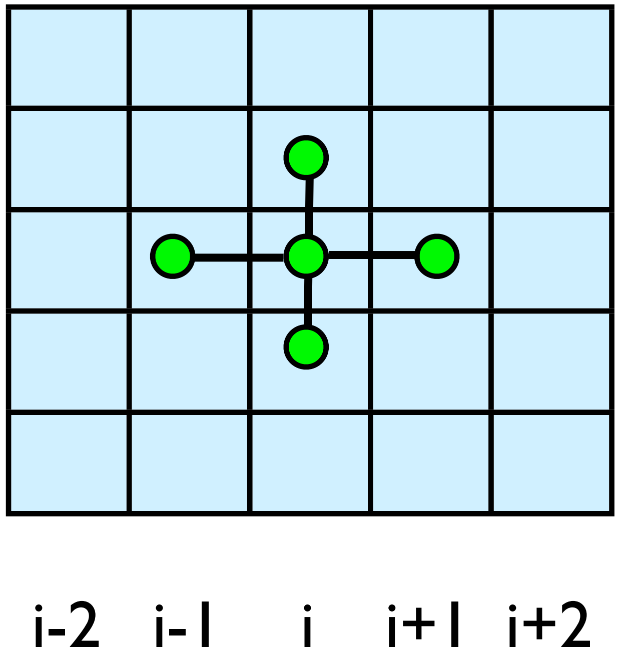
$$\left(\frac{d^2Q}{dx^2}+\frac{d^2Q}{dy^2}\right)\Bigg|_i \approx$$

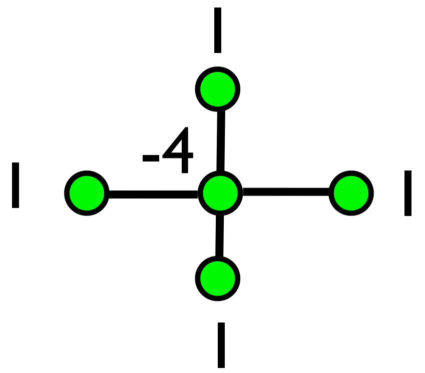
$$\frac{Q_{i+1,j+1}+Q_{i+1,j-1}-4Q_{i,j}+Q_{i-1,j+1}+Q_{i-1,j-1}}{\Delta x^2}$$

- •Done by finite differencing the discretized values

- •Implicitly or explicitly involves interpolating data and taking derivative of the interpolant

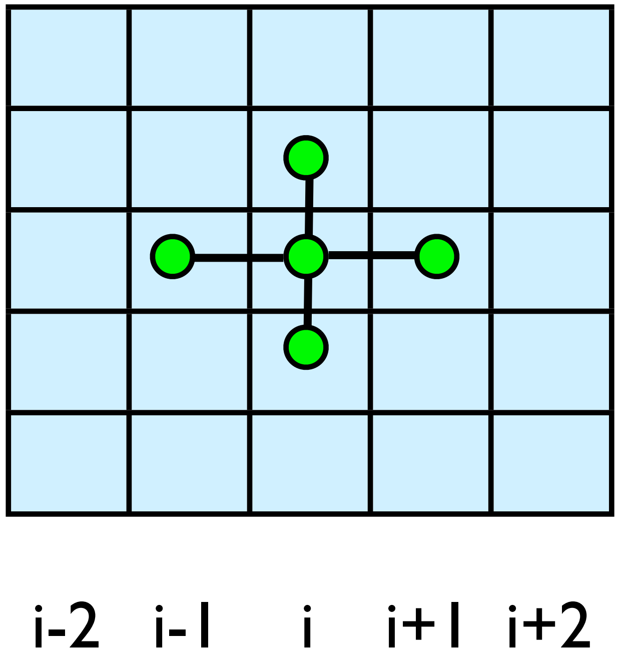
- •More accuracy - larger 'stencils'





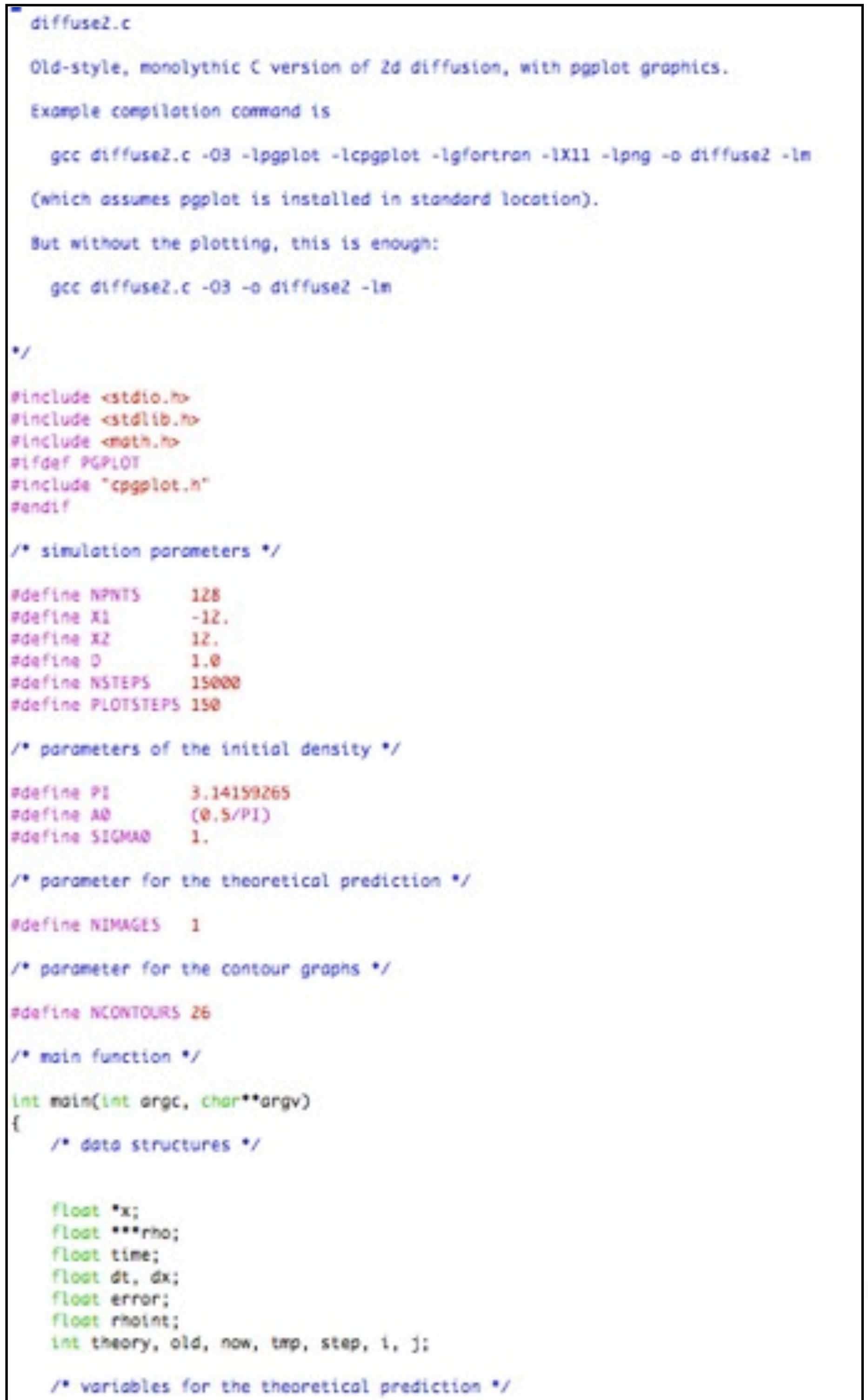
2D diffusion

- Get source code:
- https://wiki.scinethpc.ca/wiki/images/f/fb/Diffuse2.c
- Setup: Supervisor has this old code for diffusive background, “works fine for them”, wants you to add tracer particle to it.
- Uses library you don’t have for ploting - ifdef’ed out for now.



Course Project

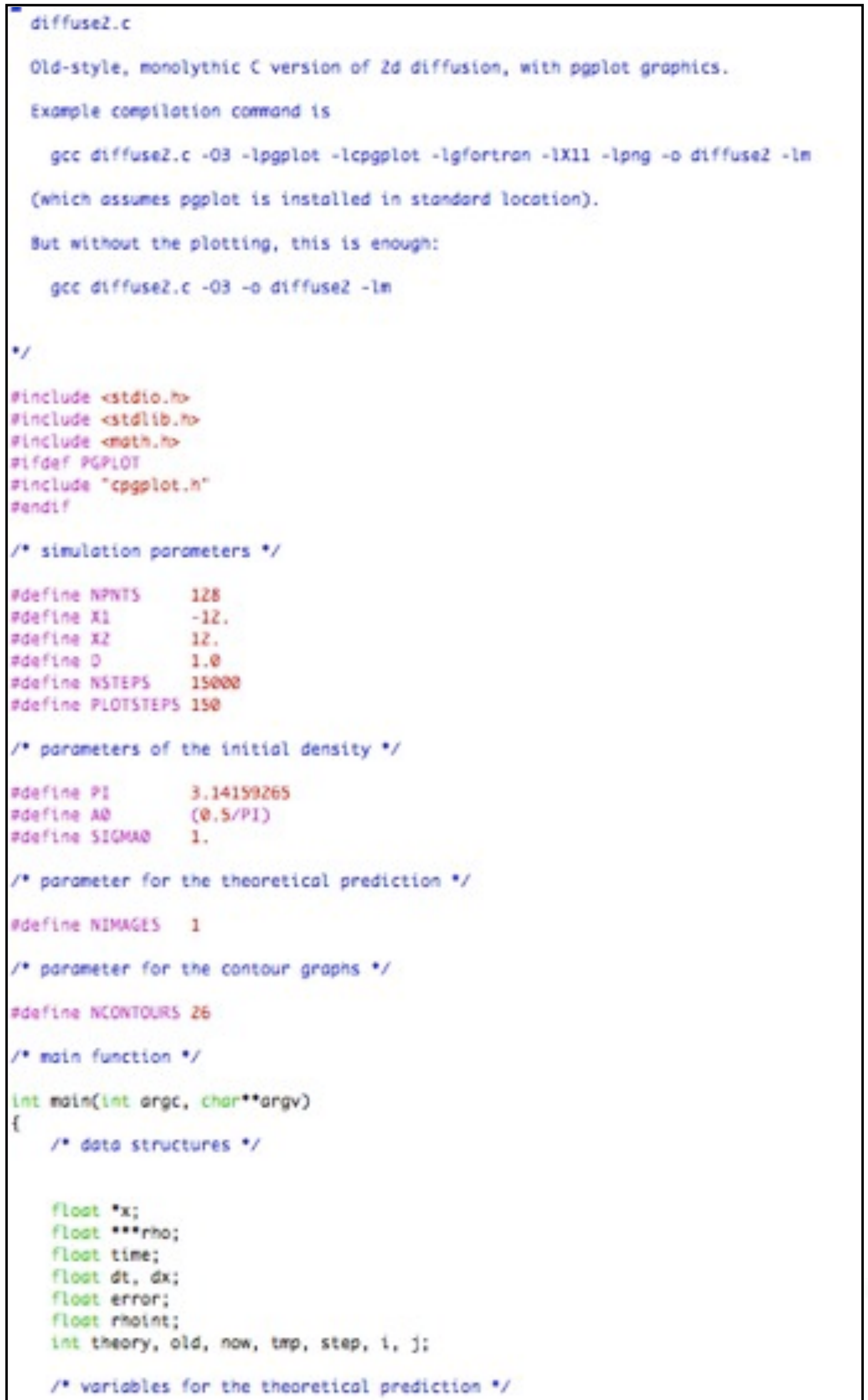
- Code isn't a disaster as these things go
- Even has comments! That are still true!
- But one monolithic routine. Difficult to follow (even in this simple 154-line case)



```
  diffuse2.c

  Old-style, monolythic C version of 2d diffusion, with pgplot graphics.

  Example compilation command is

    gcc diffuse2.c -O3 -lpgplot -lcpgplot -lgfortran -lX11 -lpng -o diffuse2 -lm

  (which assumes pgplot is installed in standard location).

  But without the plotting, this is enough:

    gcc diffuse2.c -O3 -o diffuse2 -lm

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#ifdef PGPLOT
#include "cpgplot.h"
#endif

/* simulation parameters */

#define NPNTS     128
#define X1        -12.
#define X2        12.
#define D         1.0
#define NSTEPS    15000
#define PLOTSTEPS 150

/* parameters of the initial density */

#define PI        3.14159265
#define A0        (0.5/PI)
#define SIGMA0    1.

/* parameter for the theoretical prediction */

#define NIMAGES   1

/* parameter for the contour graphs */

#define NCONTOURS 26

/* main function */

int main(int argc, char**argv)
{
    /* data structures */


    float *x;
    float ***rho;
    float time;
    float dt, dx;
    float error;
    float rhoint;
    int theory, old, now, tmp, step, i, j;

    /* variables for the theoretical prediction */
```

Course Project

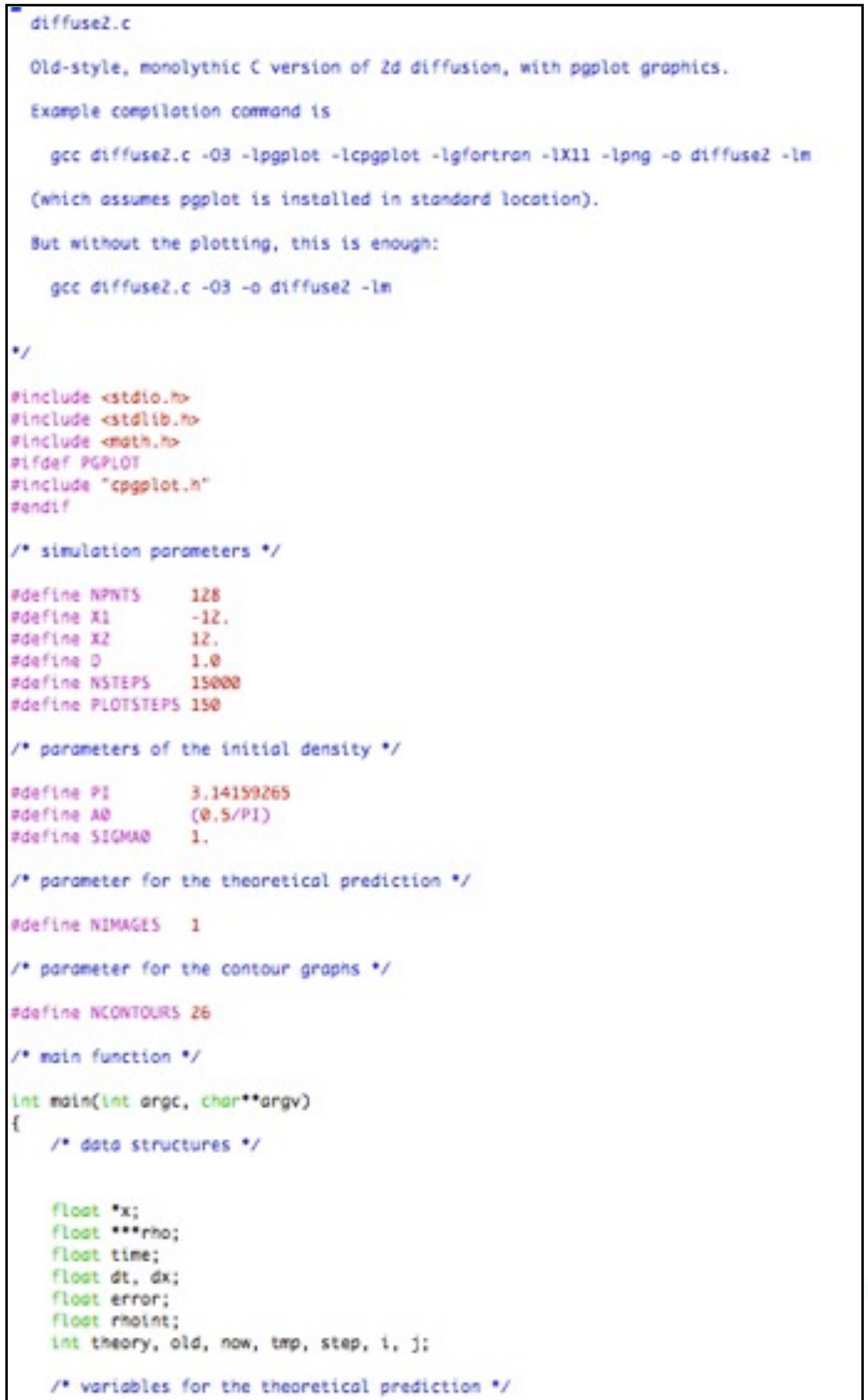
- You’re almost always better off in these situations spending some time cleaning these things up some first
- For your own sanity
- But need to make sure your changes don’t change answers
- So let’s start setting up decent development environment, baseline



```
  diffuse2.c

  Old-style, monolythic C version of 2d diffusion, with pgplot graphics.

  Example compilation command is

    gcc diffuse2.c -O3 -lpgplot -lcpgplot -lgfortran -lX11 -lpng -o diffuse2 -lm

  (which assumes pgplot is installed in standard location).

  But without the plotting, this is enough:

    gcc diffuse2.c -O3 -o diffuse2 -lm

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#ifdef PGPLOT
#include "cpgplot.h"
#endif

/* simulation parameters */

#define NPNTS     128
#define X1        -12.
#define X2        12.
#define D         1.0
#define NSTEPS    15000
#define PLOTSTEPS 150

/* parameters of the initial density */

#define PI        3.14159265
#define A0        (0.5/PI)
#define SIGMA0    1.

/* parameter for the theoretical prediction */

#define NIMAGES   1

/* parameter for the contour graphs */

#define NCONTOURS 26

/* main function */

int main(int argc, char**argv)
{
    /* data structures */


    float *x;
    float ***rho;
    float time;
    float dt, dx;
    float error;
    float rhoint;
    int theory, old, now, tmp, step, i, j;

    /* variables for the theoretical prediction */
```

Course Project

- Make a new git repository
- Start a makefile (CFLAGS=-O3 -Wall; LDFLAGS=-lm; then link line should be enough to start).
- Include a “clean” target.



```
  diffuse2.c

  Old-style, monolythic C version of 2d diffusion, with pgplot graphics.

  Example compilation command is

    gcc diffuse2.c -O3 -lpgplot -lcpgplot -lgfortran -lX11 -lpng -o diffuse2 -lm

  (which assumes pgplot is installed in standard location).

  But without the plotting, this is enough:

    gcc diffuse2.c -O3 -o diffuse2 -lm

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#ifdef PGPLOT
#include "cpgplot.h"
#endif

/* simulation parameters */

#define NPNTS     128
#define X1        -12.
#define X2        12.
#define D         1.0
#define NSTEPS    15000
#define PLOTSTEPS 150

/* parameters of the initial density */

#define PI        3.14159265
#define A0        (0.5/PI)
#define SIGMA0    1.

/* parameter for the theoretical prediction */

#define NIMAGES   1

/* parameter for the contour graphs */

#define NCONTOURS 26

/* main function */

int main(int argc, char**argv)
{
    /* data structures */


    float *x;
    float ***rho;
    float time;
    float dt, dx;
    float error;
    float rhoint;
    int theory, old, now, tmp, step, i, j;

    /* variables for the theoretical prediction */
```

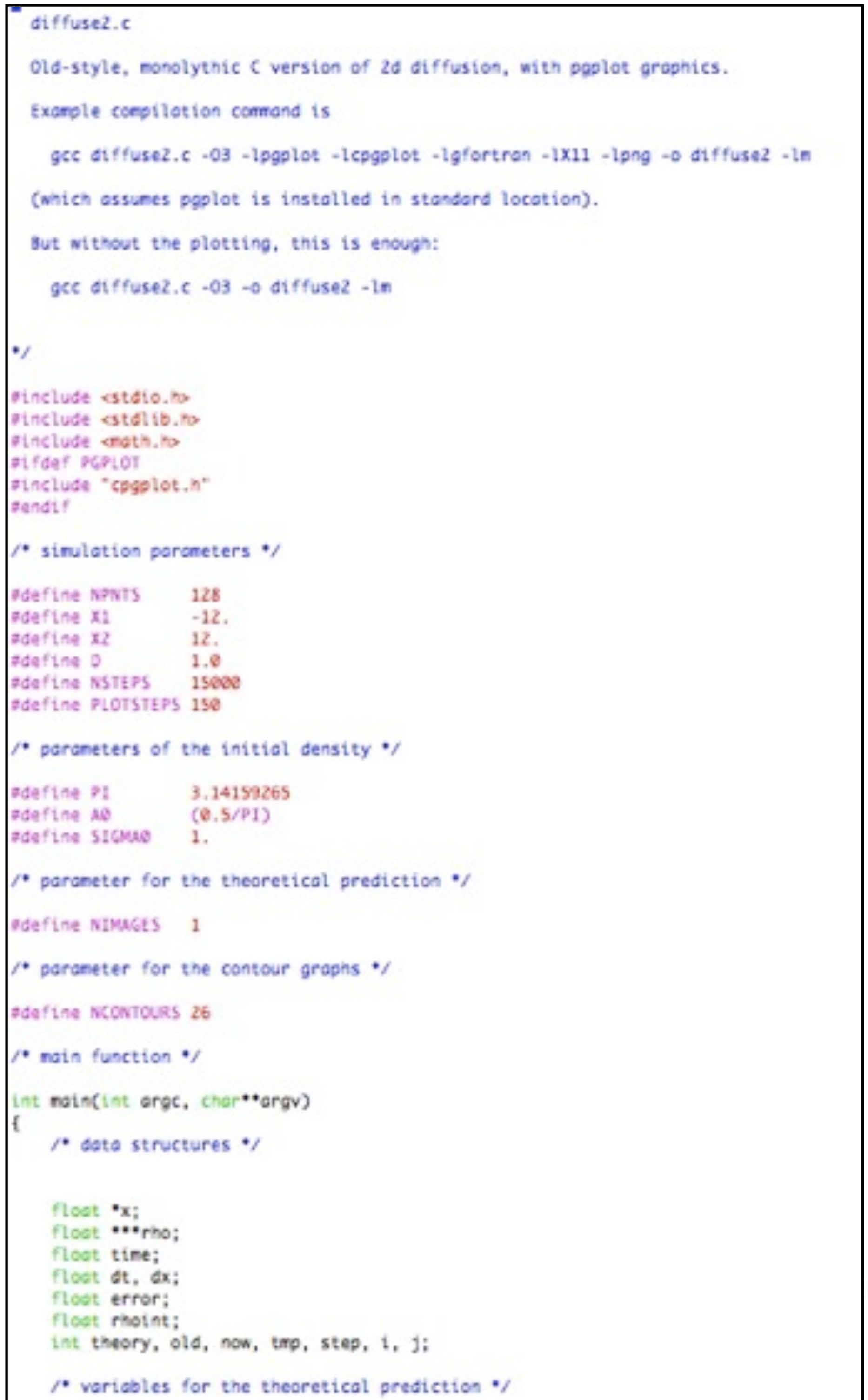
On compiler flags

- Optimization:
  - -O, -O0, -O1, -O2, -O3 ...
  - and machine/compiler specific
- -Wall

Course Project

- Make a new git repository
- Start a makefile (CFLAGS=-O3 -Wall; LDFLAGS=-lm; then link line should be enough to start).
- Include a “clean” target.

```
reposado-$ make
make: `diffuse2' is up to date.

reposado-$ ./diffuse2
Segmentation fault (core dumped)
```



```
  diffuse2.c

  Old-style, monolythic C version of 2d diffusion, with pgplot graphics.

  Example compilation command is

    gcc diffuse2.c -O3 -lpgplot -lcpgplot -lgfortran -lX11 -lpng -o diffuse2 -lm

  (which assumes pgplot is installed in standard location).

  But without the plotting, this is enough:

    gcc diffuse2.c -O3 -o diffuse2 -lm

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#ifdef PGPLOT
#include "cpgplot.h"
#endif

/* simulation parameters */

#define NPNTS     128
#define X1        -12.
#define X2        12.
#define D         1.0
#define NSTEPS    15000
#define PLOTSTEPS 150

/* parameters of the initial density */

#define PI        3.14159265
#define A0        (0.5/PI)
#define SIGMA0    1.

/* parameter for the theoretical prediction */

#define NIMAGES   1

/* parameter for the contour graphs */

#define NCONTOURS 26

/* main function */

int main(int argc, char**argv)
{
    /* data structures */


    float *x;
    float ***rho;
    float time;
    float dt, dx;
    float error;
    float rhoint;
    int theory, old, now, tmp, step, i, j;

    /* variables for the theoretical prediction */
```

Segfault - valgrind, gdb

- The more spectacular the crash, the easier to find the immediate cause.
- Segfault / Bus error - trying to access invalid regions of memory.
- Scientific codes - array bounds, pointer errors, or occasionally mis-calling a library routine

Valgrind

- Not everyone will have this
- Everyone should know about it
- Powerful tool for finding memory problems / memory access problems
- Watches every memory access.

http://valgrind.org/

-g

- Recompile with -g instead of -O3
- Keeps symbols from the source code in the executable
- Disables some optimizations; may as well disable others while we’re at it
- Allows much more information while we’re debugging.

```
$ valgrind --tool=memcheck ./diffuse2
==8930== Memcheck, a memory error detector
==8930== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==8930== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
==8930== Command: ./diffuse2
==8930==
==8930== Invalid read of size 8
==8930==    at 0x400B04: main (diffuse2.c:172)
==8930==  Address 0x4c27f20 is 0 bytes after a block of size 3,120 alloc'd
==8930==    at 0x4A0515D: malloc (vg_replace_malloc.c:195)
==8930==    by 0x400695: main (diffuse2.c:88)
==8930==
==8930== Invalid read of size 4
==8930==    at 0x400B14: main (diffuse2.c:172)
==8930==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==8930==
==8930==
==8930== Process terminating with default action of signal 11 (SIGSEGV)...

==8930==
==8930== HEAP SUMMARY:
==8930==     in use at exit: 206,464 bytes in 4 blocks
==8930==   total heap usage: 4 allocs, 0 frees, 206,464 bytes allocated
==8930==
==8930== LEAK SUMMARY:
==8930==    definitely lost: 0 bytes in 0 blocks
==8930==    indirectly lost: 0 bytes in 0 blocks
==8930==      possibly lost: 0 bytes in 0 blocks
==8930==    still reachable: 206,464 bytes in 4 blocks
==8930==         suppressed: 0 bytes in 0 blocks
```

```
    /* setup initial conditions */

    time = 0;

    for (i = 0; i < NPNTS+2; i++) {
        x[i] = X1 + (i + 0.5)*dx;
    }

    for (i = 0; i < NPNTS+2; i++) {
        for (j = 0; j < NPNTS+2; j++) {
            rho[now][i][j] = A0*exp(-(x[i]*x[i] + x[j]*x[j]) /(2.*SIGMA0*SIGMA0
        }
    }
```

```
        rhoint = 0.0;

        for (i = 0; i < NPNTS+2; i++) {
            for (j = 0; j < NPNTS+2; j++) {

               rho[now][i][j] = rho[old][i][j]
                              + dt*D/(dx*dx) *  (+rho[old][i+1][j]
                                                 +rho[old][i-1][j]
                                                 +rho[old][i][j+1]
                                                 +rho[old][i][j-1]
                                               +4*rho[old][i][j]);
               rhoint += rho[now][i][j];

            }
        }
```

Valgrind

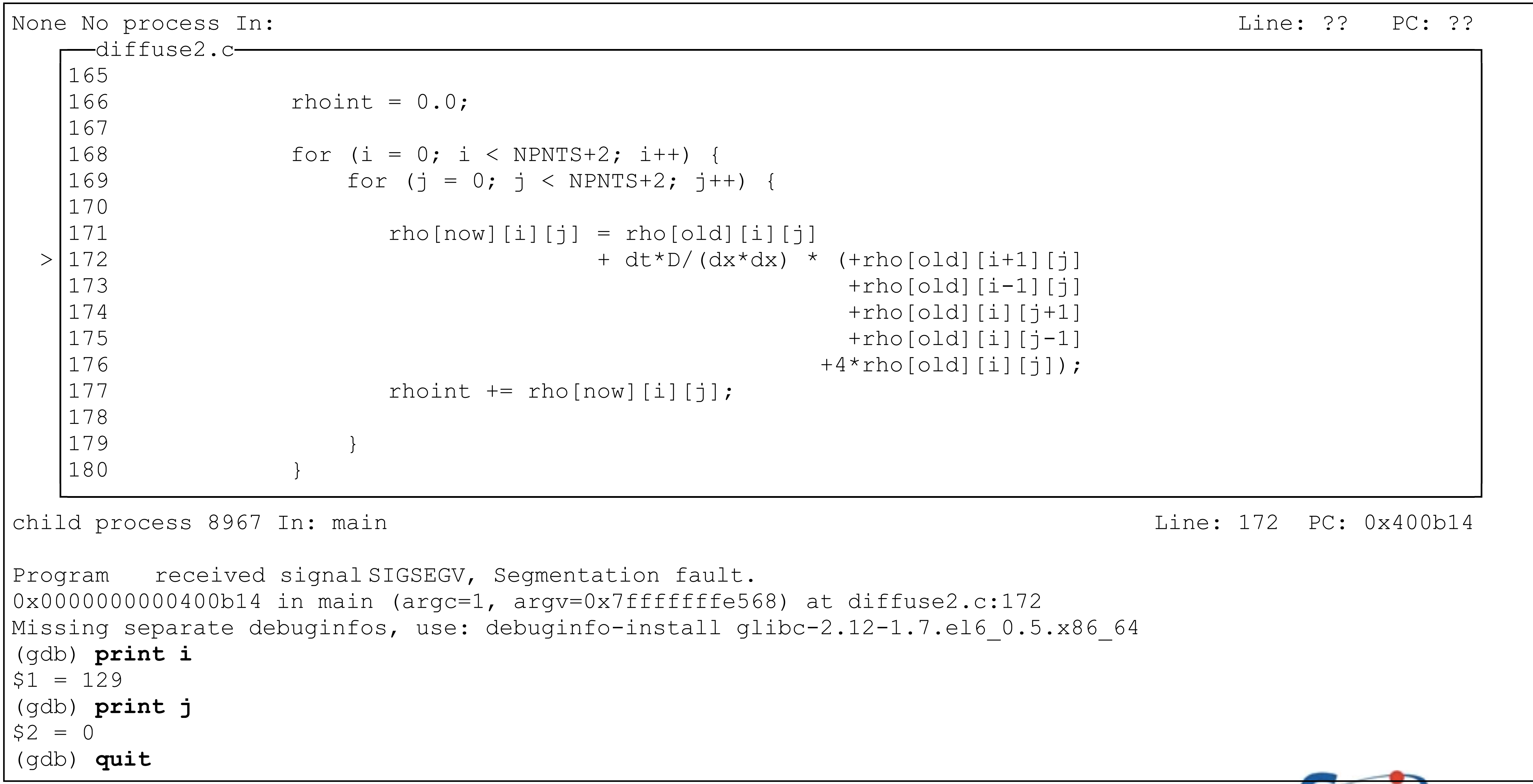
- Linux, Mac OS X
- Catches out of bounds errors, use of uninitialized variables
- Can also be used for memory **performance** problems.
- Works great for C-based languages, less well for FORTRAN

http://valgrind.org/

gdb

- Debugger; allows you to step through code one line at a time, see contents of variables, etc.
- See what code is *actually* doing as vs. what you think it is doing.
- Xcode, eclipse, visual studio - have integrated debuggers with their environments.   Principle is the same.

```
reposado$ gdb --tui ./diffuse2
run
```



```
None No process In:                                                     Line: ??   PC: ??
  ┌─diffuse2.c────────────────────────────────────────────────────────────────────────────┐
  │165                                                                                    │
  │166             rhoint = 0.0;                                                          │
  │167                                                                                    │
  │168             for (i = 0; i < NPNTS+2; i++) {                                        │
  │169                 for (j = 0; j < NPNTS+2; j++) {                                    │
  │170                                                                                    │
  │171                    rho[now][i][j] = rho[old][i][j]                                 │
 >│172                                   + dt*D/(dx*dx) * (+rho[old][i+1][j]              │
  │173                                                     +rho[old][i-1][j]              │
  │174                                                     +rho[old][i][j+1]              │
  │175                                                     +rho[old][i][j-1]              │
  │176                                                   +4*rho[old][i][j]);              │
  │177                    rhoint += rho[now][i][j];                                       │
  │178                                                                                    │
  │179                 }                                                                  │
  │180             }                                                                      │
  └───────────────────────────────────────────────────────────────────────────────────────┘
child process 8967 In: main                                          Line: 172   PC: 0x400b14

Program   received signal SIGSEGV, Segmentation fault.
0x0000000000400b14 in main (argc=1, argv=0x7fffffffe568) at diffuse2.c:172
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.7.el6_0.5.x86_64
(gdb) print i
$1 = 129
(gdb) print j
$2 = 0
(gdb) quit
```

Fix this bug

- So let's fix this classic indexing bug and recompile

But problems remain...

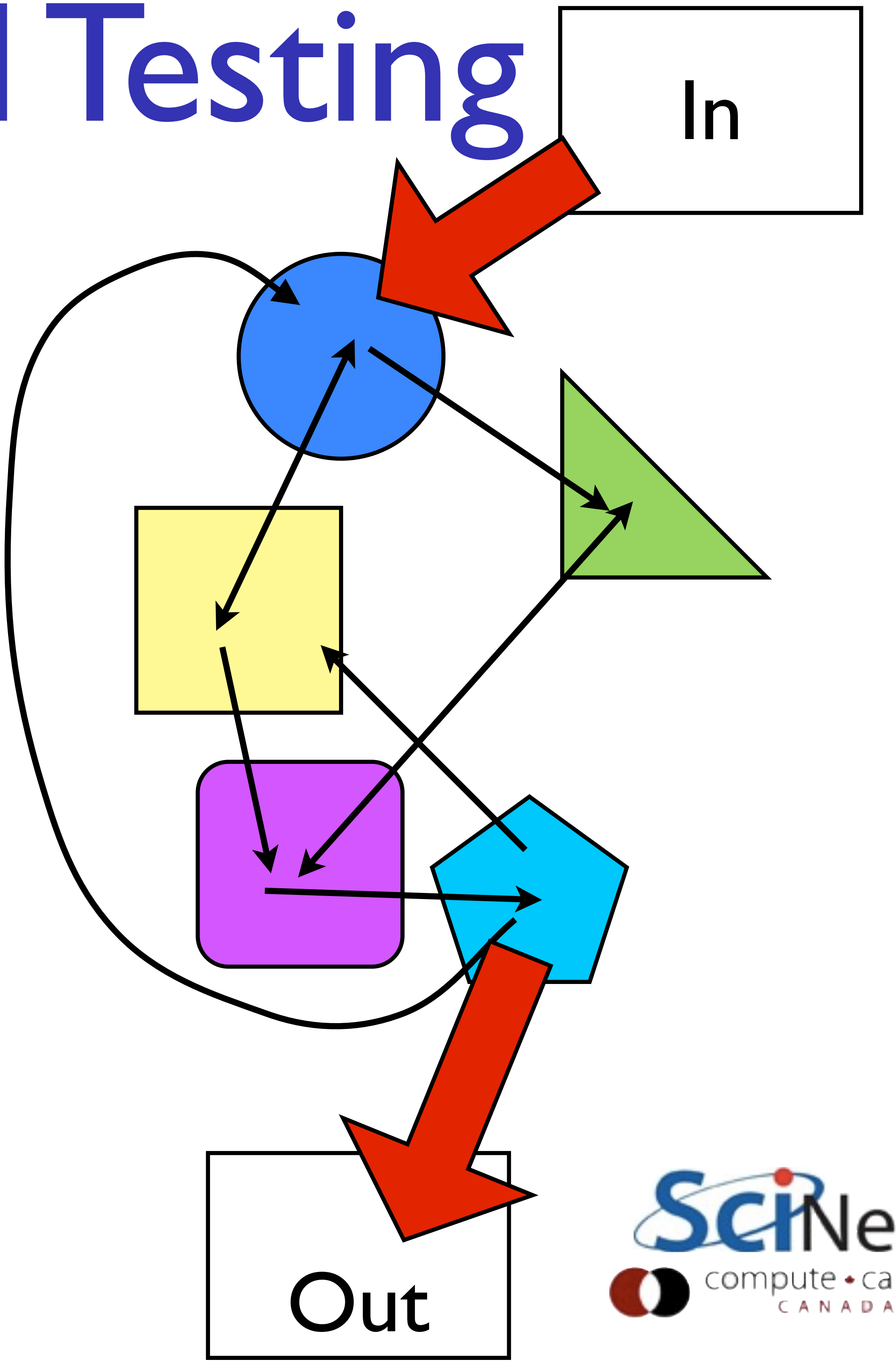
```
$ ./diffuse2 | more
Step = 0, Time = 0.00714241, Error = 2.38837, Integrated density = 2.599993
Step = 1, Time = 0.0142848, Error = 2.38837, Integrated density = 6.759978
Step = 2, Time = 0.0214272, Error = 2.38837, Integrated density = 17.575960
Step = 3, Time = 0.0285697, Error = 2.38837, Integrated density = 45.697430
Step = 4, Time = 0.0357121, Error = 2.38837, Integrated density = 118.813644
Step = 5, Time = 0.0428545, Error = 2.38837, Integrated density = 308.915619
Step = 6, Time = 0.0499969, Error = 2.38837, Integrated density = 803.178406
Step = 7, Time = 0.0571393, Error = 2.38837, Integrated density = 2088.266357
Step = 8, Time = 0.0642817, Error = 2.38837, Integrated density = 5429.500977
....
Step = 89, Time = 0.642818, Error = 2.38837, Integrated density = inf
Step = 90, Time = 0.64996, Error = 2.38837, Integrated density = inf
Step = 91, Time = 0.657103, Error = 2.38837, Integrated density = inf
```

Testing

- Crashes are easy to find (although sometimes harder to find root cause of)
- Wrong answers are harder
- **Slightly** wrong answers hardest of all (but most dangerous!)

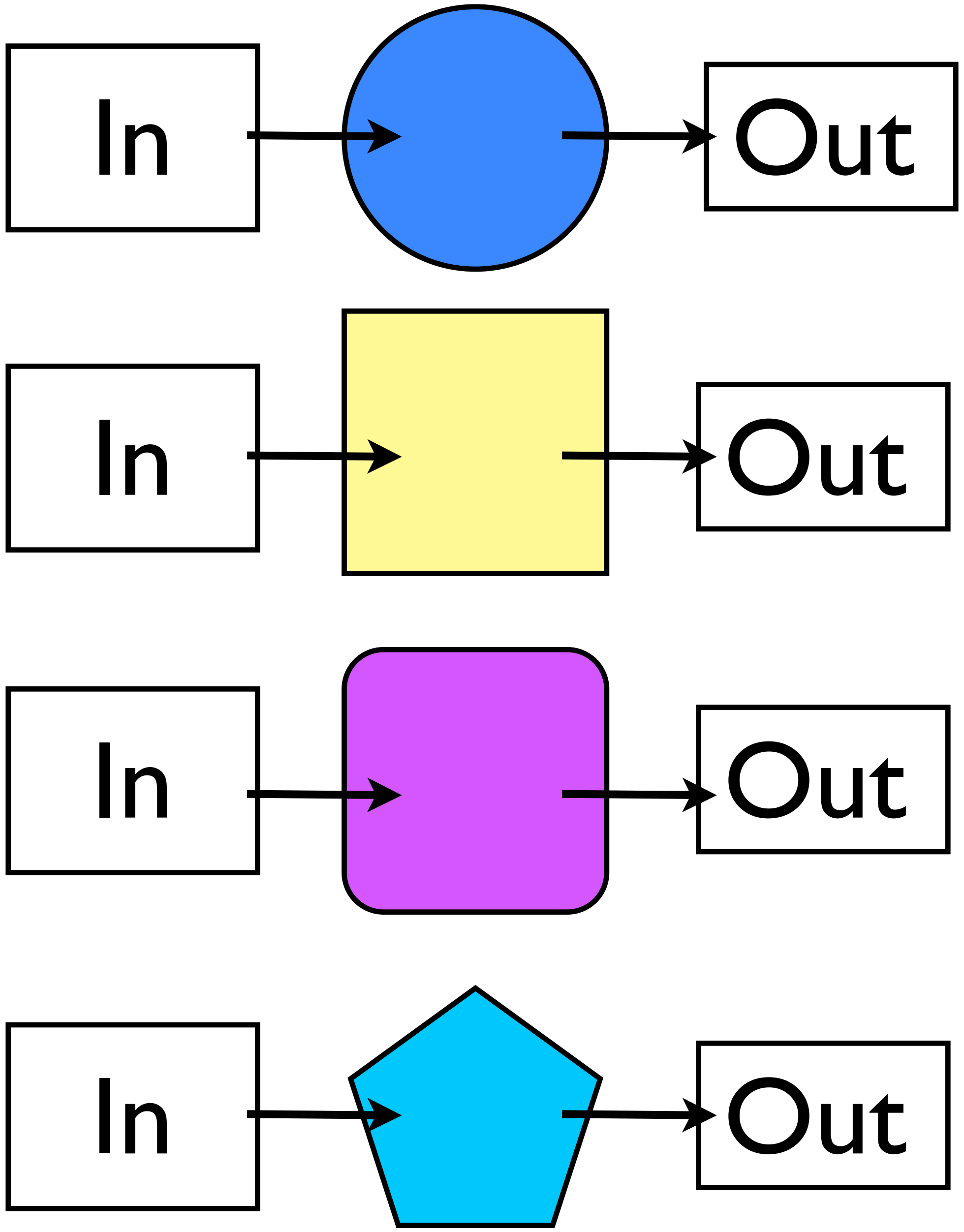
Integrated Testing

- Complicated piece of software, with many interacting parts
- Difficult to tell where a problem begins in final answer
- Integrated testing



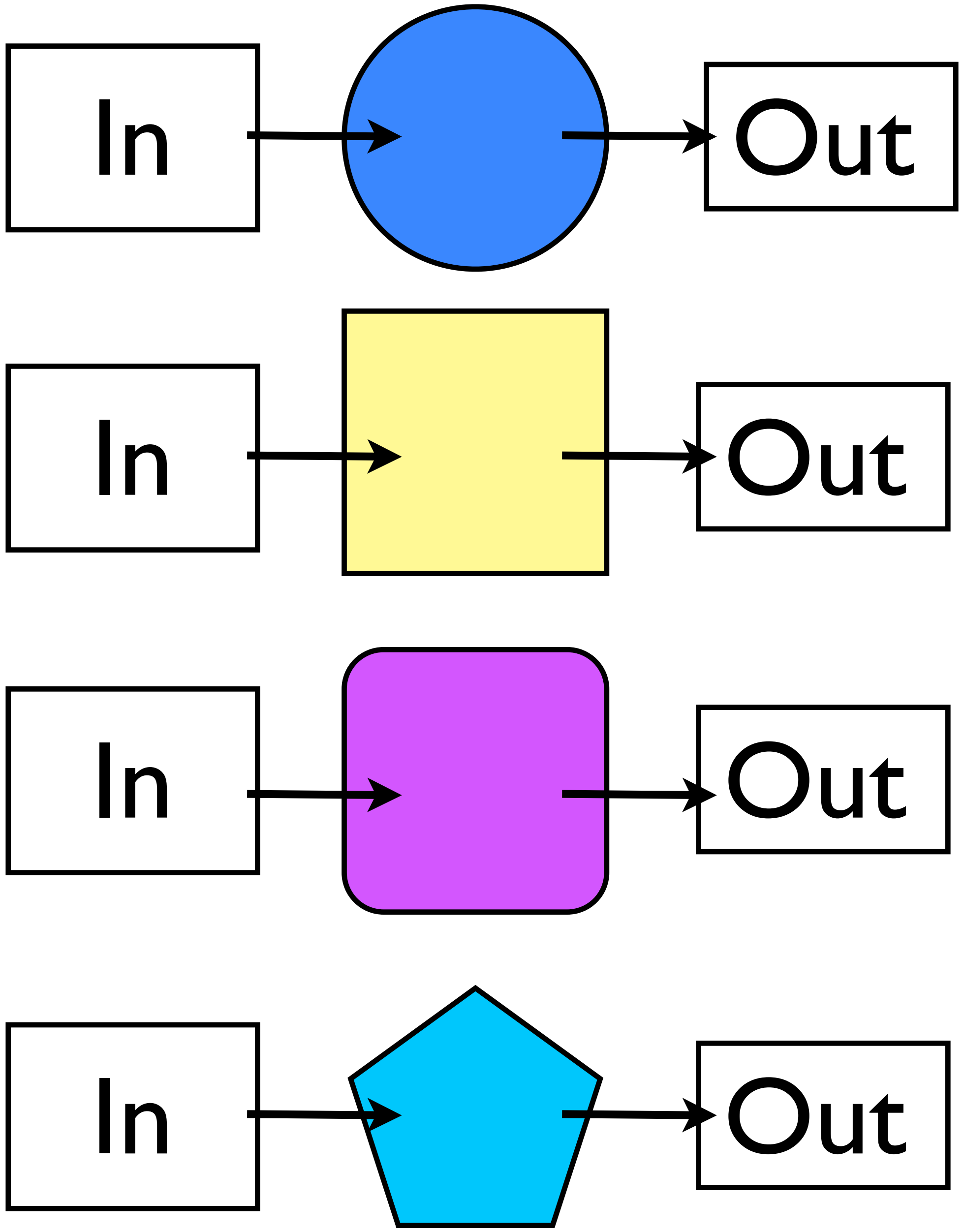
Unit Testing

- Testing major pieces of the code individually
- Comparing easy solutions, "typical" solutions, wierd edge cases
- Enormously speeds up, simplifies, finding problems when introduced



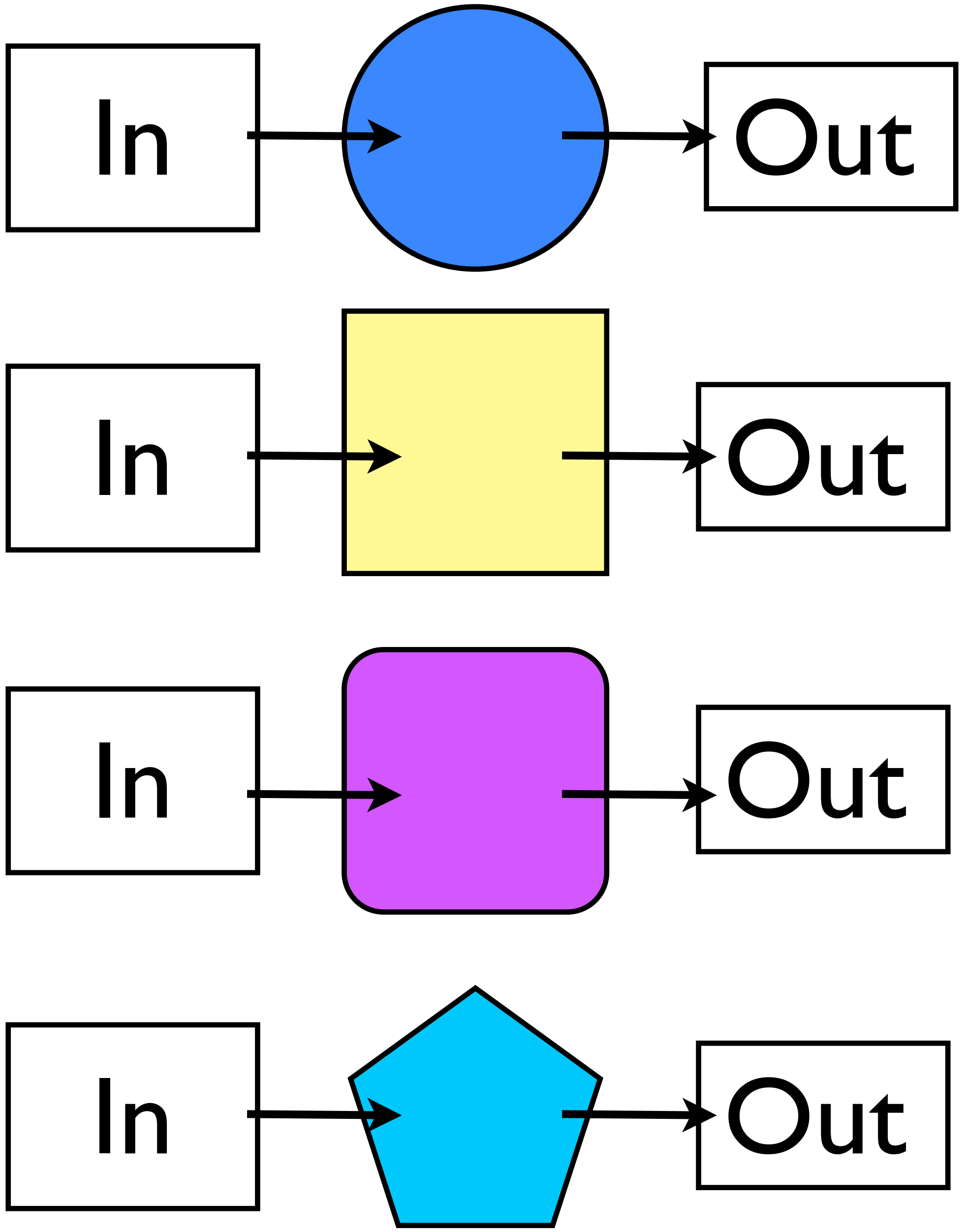
Testing

- Complex piece of software which **doesn’t** have testing regularly done on it - integrated and unit?
- You can save yourself a lot of time and just assume it’s wrong.



Testing and Modularity

- Modular software is needed for unit testing
- Have to have separable, independant units.
- Also answers the question "how much should be in module" - what would be good independant tests?



Testing Frameworks

- There are lots of excellent testing frameworks that you can use - Google Tests (C++), xUnit, Check (C), Nose (python), JUnit (Java)
- They're great, but they have a big learning curve.
- You don't need anything that elaborate to get started with unit testing.

diffusionOperator.c

```
int diffusionOperator(float **rhoOld,   /* original field */
        const int n, const int m,       /* size of interior grid */
        float dt, float dx, float D,    /* parameters of diffusion */
        float **rhoNew, float *rhoint ) /* outputs */
{

    /* code goes here... */

    return 0;
}


int testDiffusionOperatorConstant() {

    /* give it one field and test its answer */

}

int testDiffusionOperatorGradient() {

    /* give it one field and test its answer */

}

int runDiffusionOperatorTests() {
    /* run each of the tests */
}
```

diffusionOperatorTests.c

```
int main() {

   int runDiffusionOperatorTests();

}
```

Makfile

```
...
diffusionOperatorTests: diffusionOperatorTests.o diffusionOperator.o
   $(CC) -o $@ $^ $(LDFLAGS)
```