Scientific Computing (Phys 2109/ Ast 3100H)

I. Scientfic Software Development

SciNet HPC Consortium

SciNet compute • calcul CANADA

HW3 Issues

- How to do the timesteping (in what sense does the coupling operate?)
- Tracer Theory doesn't know about diffusion?

Makefile: multiple executables

- Convention, like with "clean", is an "all" target which lists all targets
- Make everything at once:

```
CC=gcc
CFLAGS=...

all: main interpolation_test

main: ....

interpolation_test: ....

clean:
   ...
```

Makefile:
Run tests

- Easier it is to run tests, more often you test
- Can make a target (typically, 'check') which **runs** any/all tests you have:

```
CC=gcc
CFLAGS=...

all: main interpolation_test

check: interpolation_test
   ./interpolation_test

main: ....

interpolation_test: ....
```

SciNet compute • calcul CANADA

Development Tools

- By HW3, code is already starting to get nontrivial
- Coupled physics - 400-900 lines of interacting C code
- Sometimes non-obvious decisions needed to be made about where one module stopped and one started
- Without modularity, would be a tangled, un-debugable mess

Development Tools

- Going to talk a little more about some tools to help you develop faster/smarter/better
- One is already mentioned - debugger (gdb)
- One is for software performance - profiler (gprof)
- Other debuggers, profilers similar

SciNet compute • calcul CANADA

Debuggers

- Debugging is the worst thing in the world.
- Time-sucking, demoralizing, horrible, slow, awkward, waste of time.
- Worse than that, really.
- We'll talk about using gdb or similar debuggers, but first want to give a crash course on debugging

SciNet
compute • calcul
CANADA

Debuggers

- Debugging is something you want to (in this order!):
  - Avoid
  - Make easier
  - Get good at -- which means finding the problem quickly, and solving it correctly

SciNet compute • calcul CANADA

Write Code Professionally

- Even a `one off' program will hang around for a long time.
- ALWAYS
- Make assumptions explicit
- Test for them (`assert()`)
- Test for error conditions/return codes
- Write modular code

SciNet compute • calcul CANADA

Defensive Programming

- Fail early, fail often.
- If your function assumes n>0, or only works for even input, or..
- Document it
- Test for it

```
if (n<0) {
      fprintf(stderr,"Error in myfunc(); ");
      fprintf(stderr," n = %d <0 !\n",n);
      return;
}
```

```
#include <assert.h>
.
.
assert(n>=0);
```

SciNet compute • calcul CANADA

Test for errors

- Related to making your assumptions explicit
- Did you:
  - make sure `malloc()` didn't return NULL?
  - Make sure you aren't at `EOF`?
  - Make sure `fopen()` opened the file?
  - Make sure scanf() read in the right number of variables?

Test your code

- For every non-trivial section of your code, write tests
  - And then run them regularly!
- Much easier with properly modular code
- Test special cases (passed in too small an array) and real cases.

Compiler is your friend

- Always turn on warnings in compiler
- -Wall, -warn all
- And fix all the things it complains about!
- Have debugging flags turned on during development - eg, -fcheck=bounds in gfortran

All of this takes time..

- But much, much less time than debugging
- Debugging - hours of work spent just to get the code back into the shape it *should* have been in the first place.
- Not a journey of personal growth - just a collossal waste of time.
- Avoid it.

But a bug happened anyway...

- Five steps to debugging:
  - Find out you have a bug
  - Find out roughly where that bug is
  - Examine section of code
  - Find *and understand* the problem, and any related problems
  - Fix it.

SciNet compute • calcul CANADA

A futile slide?

Don't ever, ever see a bug and immediately rush in with the editor to change things until it goes away.

Really only learn this the hard way - night before a deadline, need to `save time' by leaping in and fixing things.

Life lesson: 10+ hours of panicked programming can often save you from one hour of careful thought.

SciNet compute • calcul CANADA

Finding you have a bug

- Test, test, test.
- Only thing worse than debugging is not realizing you have a bug.
- Test with invalid inputs, valid inputs, etc.
- Once you know you have a bug, find **simplest, fastest** case which demonstrates the bug

Finding where bug is

- Need to bisect.
- Figure out where wrong answer happens.
- Look at results half way through computation. Is it wrong?
  - If so, problem is in first half
  - Otherwise, in second half
- Repeat

SciNet compute • calcul CANADA

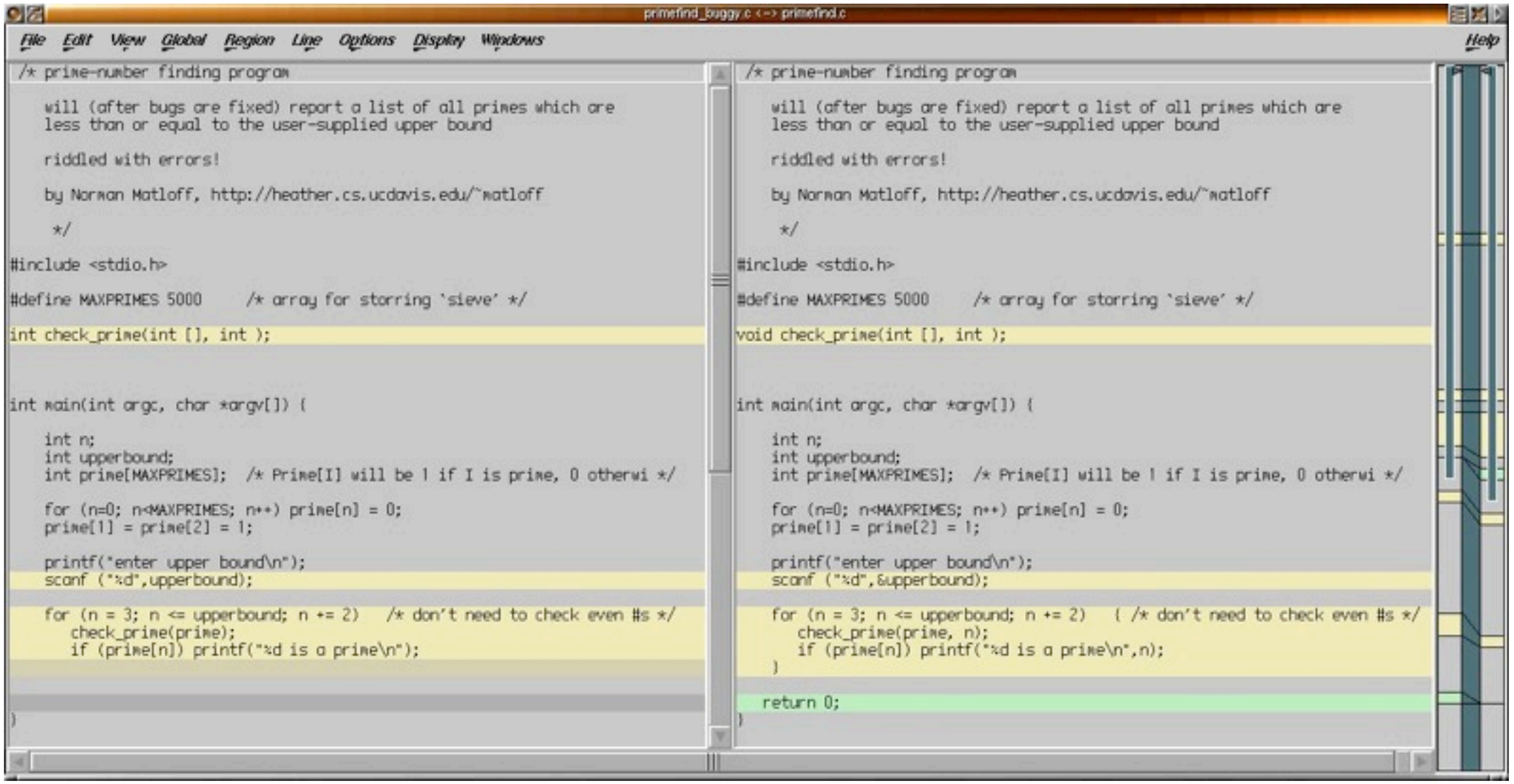
Git is your friend

- If you have version control, and had a known working version earlier, can hugely simplify finding bug.
- If know when bug was introduced, know what changes were made!
- Greatly reduces lines of code you're sifting through.

Git is your friend

- git diff will show you all changes between your last commit and current files
- (Look into git difftool for nice graphical version)
- You can use git reset --hard or similar to revert back to working version
- If correct version was much further back and you want to find where problem happened, “git bisect” will help you find where problem was introduced.

Graphical diffs



- tkdiff, xxdiff, kompare, etc - will change your life.

SciNet compute • calcul CANADA

Finding bug: debugger

- Typical scientist way of finding bugs - inserting printf()s into code.
- Slow, error-prone, and only shows what you print.
- Much better - debugger.
- We'll walk through gdb (lowest common denominator).
- Graphical debuggers - Linux, ddd, eclipse, ddt; Mac - Xcode; Windows - Visual studio

SciNet
compute • calcul
CANADA

gdb hands-on

- Edit your HW3 makefile and add -g to compile, link line
- Includes information in the executable about source code
- Disables some optimizations
- make clean, then make
- gdb [programname]

SciNet compute • calcul CANADA

Running program

- at (gdb) prompt, type “run”
- Runs program as usual (slightly slower)
- If program required command-line arguments, “set args [args]”, then run

```
$ gdb ./main
GNU gdb 6.3.50-20050815 (Apple version gd
Copyright 2004 Free Software Foundation,
GDB is free software, covered by the GNU
welcome to change it and/or distribute co
Type "show copying" to see the conditions
There is absolutely no warranty for GDB.
This GDB was configured as "x86_64-apple-
done

(gdb) run
.....

Program exited normally.
(gdb)
```

SciNet compute • calcul CANADA

Breakpoints

- Insert a breakpoint into the code
- Program will run until it hits breakpoint, then stop
- can use function name, or filename:line number

```
$ gdb ./main
GNU gdb 6.3.50-20050815 (Apple version gdb-1
Copyright 2004 Free Software Foundation, Inc
GDB is free software, covered by the GNU Gene
welcome to change it and/or distribute copies
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Typ
This GDB was configured as "x86_64-apple-darw
done

(gdb) run
.....

Program exited normally.
(gdb) break tracer_compute_force_and_velocity
Breakpoint 1 at 0x100002238: file tracer.c, l
```

SciNet compute • calcul CANADA

Breakpoints

- Put breakpoint at tracer_compute_force_and_velocity
- Then run
- “list” lists the next few lines of code (you can “list 25” to list starting at 25)
- print c - why is it not 0?

```
(gdb) break tracer_compute_force_and_velocity
Breakpoint 1 at 0x100002238: file tracer.c, l

Starting program: /Users/ljdursi/Desktop/NovC
0.000000 0.000000 0.000000

Breakpoint 1, tracer_compute_force_and_velocit
35      for (int c=0; c<DIM; ++c) {
```

step/next

- Can step through code, looking at results
- Much more powerful than printf
- step - do next line of code regardless of where it is
- next - next line of code in this routine (eg, step over function calls)

```
(gdb) list
30        tracer->alpha = tracer->alpha0*(1
tracer->r[0], tracer->r[1]));
31    }
32
33    void tracer_compute_force_and_velocit
34    {
35        for (int c=0; c<DIM; ++c) {
36            tracer->v[c] = (tracer->r[c]
37            tracer->f[c] = -tracer->alpha
38        }
39        tracer->f[0] += tracer->qE;
(gdb) step
36            tracer->v[c] = (tracer->r[c]
(gdb) step
37            tracer->f[c] = -tracer->alpha
(gdb) print c
$2 = 0
(gdb) print *tracer
$3 = {
  m = 1,
  x1 = 0,
  x2 = 10,
  qE = 1,
```

Conditional breakpoints

- Delete previous breakpoint (delete 1)
- Let's say we want to examine periodic boundaries
- Put a breakpoint in tracer_timestep on line where L is calculated if need periodic in x direction
- cont continues run

```
(gdb) delete 1
(gdb) break tracer.c:51 if tracer->r[0] >
      tracer->x2 || tracer->r[0]
      < tracer->x1
Breakpoint 2 at 0x1000015e3: file tracer.
(gdb) cont
...
Breakpoint 2, tracer_time_step (tracer=0x
51      float L = tracer->x2 - tracer->x1
```

SciNet compute • calcul CANADA

Conditional breakpoints

- When stepping through, you can set variables:
- eg, 'set var L=5.'
- See if that alters behaviour...

```
(gdb) delete 1
(gdb) break tracer.c:51 if tracer->r[0] >
      tracer->x2 || tracer->r[0]
      < tracer->x1
Breakpoint 2 at 0x1000015e3: file tracer.
(gdb) run
...
Breakpoint 2, tracer_time_step (tracer=0x
51      float L = tracer->x2 - tracer->x1
```

Watchpoints

- Can break anywhere in program if a variable changes
- Very useful for tracking changes to a variable you think should be same!
- delete (deletes all breakpoints)
- break main; run

```
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) break main
Breakpoint 3 at 0x1000009b4: file main.c,
(gdb) run
Breakpoint 2, main () at main.c:12
12        float m  = 1.0;
```

SciNet compute • calcul CANADA

Watchpoints

- Set a watchpoint for your tracer particle's r[0]
- continue
- Should stop in tracer_init

```
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) break main
Breakpoint 3 at 0x1000009b4: file main.c,
(gdb) run
Breakpoint 2, main () at main.c:12
12        float m  = 1.0;
(gdb) watch tracer->r[0]
Hardware watchpoint 3: tracer->r[0]
(gdb) cont
Continuing.
Hardware watchpoint 3: tracer->r[0]

Old value = 4.59163468e-41
New value = 0
tracer_init (tracer=0x7fff5fbff600, r0=0x
22          tracer->rprev[c] = r0[c] - v0
```

SciNet compute • calcul CANADA

Stack frame

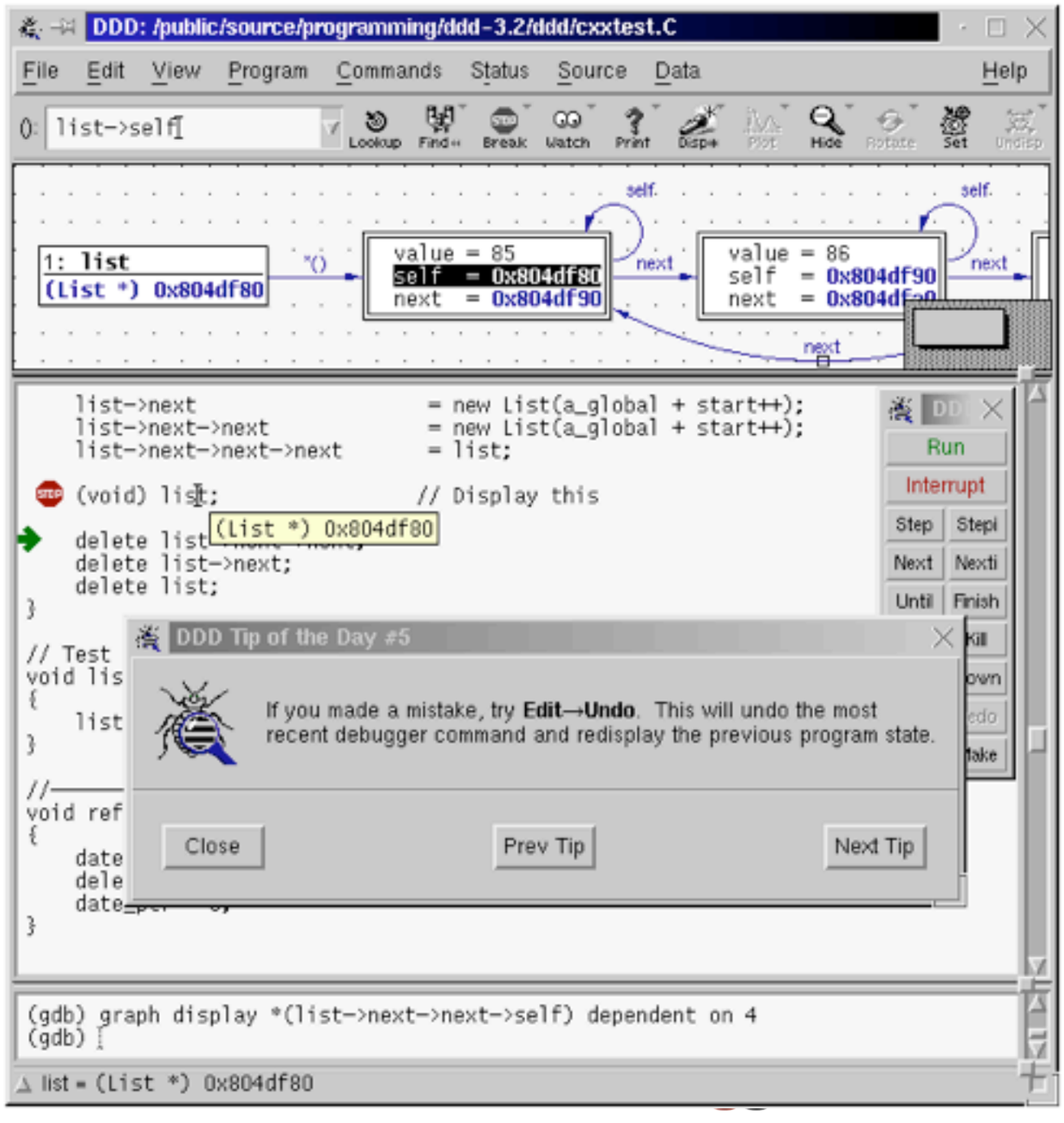
- where - Shows where you are in the code and how you got there
- eg, line 68 of main.c called tracer_init at line 22
- Can go between callees and callers with “frame” command
- Lets you see values in caller, undersand why it was called with values it was.

```
Old value = 4.59163468e-41
New value = 0
tracer_init (tracer=0x7fff5fbff600, r0=0x7fff5f
22          tracer->rprev[c] = r0[c] - v0[c]*tr

(gdb) where
#0  tracer_init (tracer=0x7fff5fbff600, r0=0x7f
#1  0x0000000100000e16 in main () at main.c:68
(gdb) frame 1
#1  0x0000000100000e16 in main () at main.c:68
68      tracer_init(&tracer,r0,v0);
(gdb) print r0
$1 = {0, 0}
(gdb) print v0
$2 = {0, 10}
(gdb) print tracer
$3 = {
  m = 1,
  x1 = 0,
  x2 = 10,
```

SciNet compute • calcul CANADA

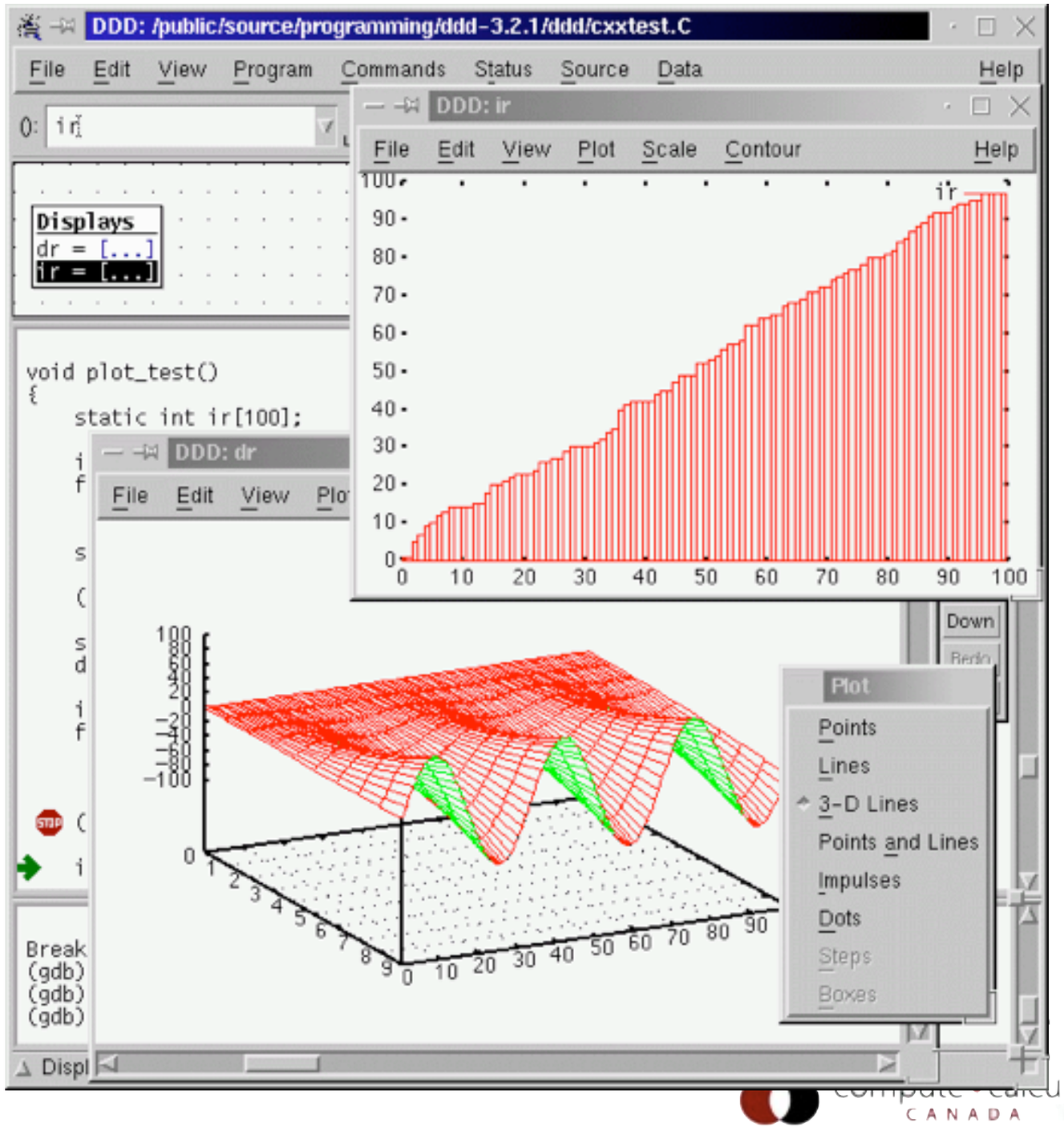
Graphical debuggers

- Show source all at once, allow you to manipulate/view data in other windows
- Easier to use
- Same basic functionality



ddd

- ddd (http://www.gnu.org/s/ddd/) is particularly nice for scientists:
  - free
  - Built-in plotting of arrays

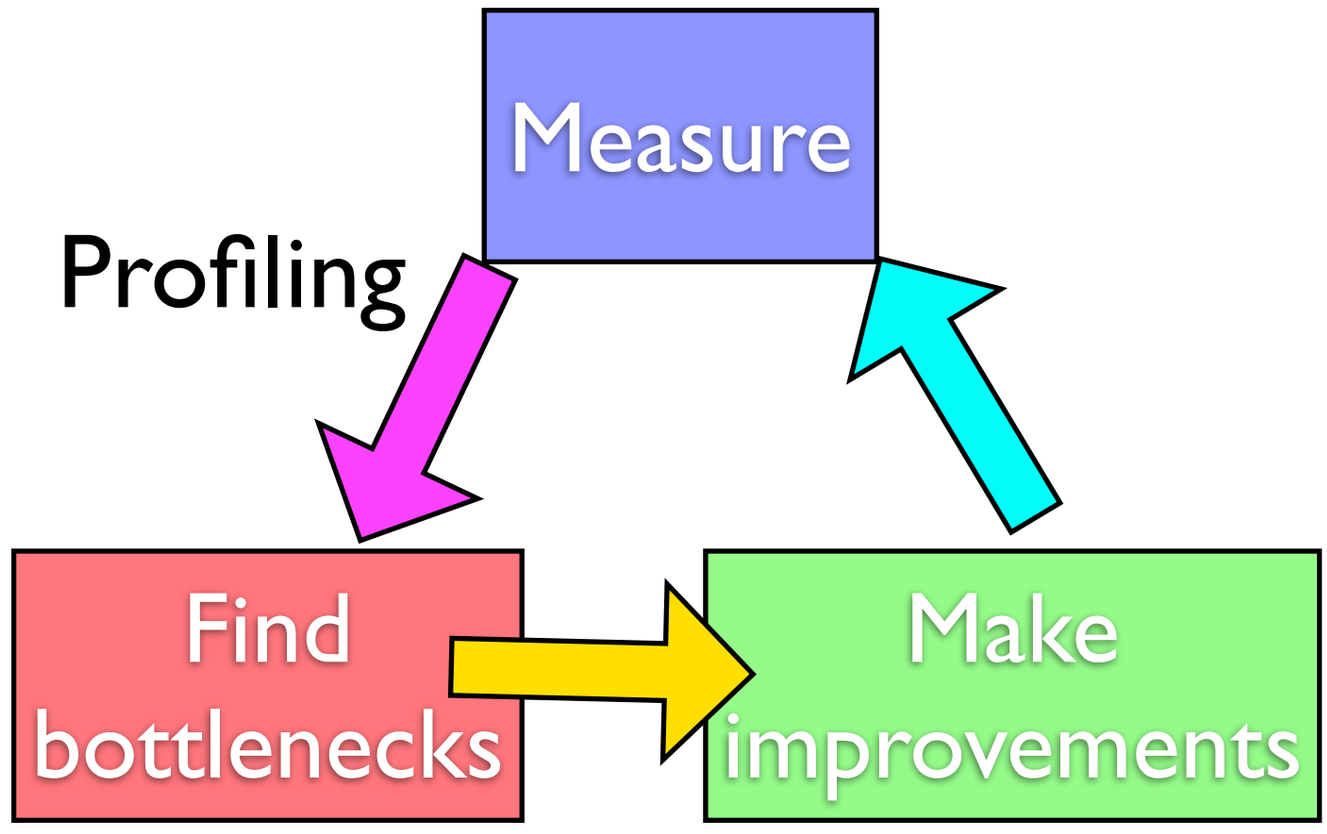


Performance

- Get things right before getting them fast
- **Once** your program is working correctly (and has tests so you can make sure it's still working correctly after changes), can work on making it go faster

How to improve Performance?

- Can't improve what you don't measure
- Have to be able to quantify where your problem spends its time.



SciNet compute • calcul CANADA

Profiling A Code

- Where in your program is time being spent?
- Find the expensive parts
  - Don't waste time optimizing parts that don't matter

```
        case SIM_PROJECTILE:
            ymin = xmin = 0.;
            ymax = xmax = 1.;
            dx = (xmax-xmin)/npts;
            dy = (ymax-ymin)/npts;
            init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
            projectile_initvalues(&d, psize, pdens, pvel);
            outputvar = DENSVAR;
            break;
    }

    /* apply boundary conditions and make thermodynamically consistant */
    bcs[0] = xbc; bcs[1] = xbc;
    bcs[2] = ybc; bcs[3] = ybc;
    apply_all_bcs(&d,bcs);
    domain_backward_dp_eos(&d);
    domain_ener_internal_to_tot(&d);


    /* main loop */

    tick(&tt);
    if (output) domain_plot(&d);
    printf("Step\tdt\ttime\n");
    for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

        printf("%d\t%g\t%g\n", step, dt, time);

        if (output && ((step % outevery) == 0) ) {
            sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
            sprintf(binfilename,"dens_test_%d.bin", outnum);
            sprintf(h5filename,"dens_test_%d.h5", outnum);
            sprintf(ncdffilename,"dens_test_%d.nc", outnum);
            domain_output_ppm(&d, outputvar, ppmfilename);
            domain_output_bin(&d, binfilename);
            domain_output_hdf5(&d, h5filename);
            domain_output_netcdf(&d, ncdffilename);
            domain_plot(&d);
            outnum++;
        }
        kl_timestep_xy(&d, bcs, dt);
        apply_all_bcs(&d,bcs);

        kl_timestep_yx(&d, bcs, dt);
        apply_all_bcs(&d,bcs);
    }
    tock(&tt);
```

SciNet compute • calcul CANADA

Profiling A Code

- Timing *vs.* Sampling *vs.* Tracing
- Instrumenting the code vs. Instrumentation-free

```
        case SIM_PROJECTILE:
            ymin = xmin = 0.;
            ymax = xmax = 1.;
            dx = (xmax-xmin)/npts;
            dy = (ymax-ymin)/npts;
            init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
            projectile_initvalues(&d, psize, pdens, pvel);
            outputvar = DENSVAR;
            break;
    }

    /* apply boundary conditions and make thermodynamically consistant */
    bcs[0] = xbc; bcs[1] = xbc;
    bcs[2] = ybc; bcs[3] = ybc;
    apply_all_bcs(&d,bcs);
    domain_backward_dp_eos(&d);
    domain_ener_internal_to_tot(&d);


    /* main loop */

    tick(&tt);
    if (output) domain_plot(&d);
    printf("Step\tdt\ttime\n");
    for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

        printf("%d\t%g\t%g\n", step, dt, time);

        if (output && ((step % outevery) == 0) ) {
            sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
            sprintf(binfilename,"dens_test_%d.bin", outnum);
            sprintf(h5filename,"dens_test_%d.h5", outnum);
            sprintf(ncdffilename,"dens_test_%d.nc", outnum);
            domain_output_ppm(&d, outputvar, ppmfilename);
            domain_output_bin(&d, binfilename);
            domain_output_hdf5(&d, h5filename);
            domain_output_netcdf(&d, ncdffilename);
            domain_plot(&d);
            outnum++;
        }
        kl_timestep_xy(&d, bcs, dt);
        apply_all_bcs(&d,bcs);

        kl_timestep_yx(&d, bcs, dt);
        apply_all_bcs(&d,bcs);
    }
    tock(&tt);
```

SciNet compute • calcul CANADA

Timing whole program

- Very simple; can run any command
- In serial, real = user+sys
- In parallel, ideally user = (nprocs)x (real)

```
$ time ./a.out

[ your job output ]

real    0m2.448s
user    0m2.383s
sys     0m0.027s
```

Elapsed "walltime"

Actual user time

System time: Disk, I/O...

SciNet compute • calcul CANADA

Running "top"

- Run top in another terminal window while program runs
- Tells CPU usage, amount of memory, status
- Lots of D/S status - waiting for I/O

```
Processes: 128 total, 5 running, 2 stuck, 121 sleeping, 477 threads
Load Avg: 2.34, 2.28, 2.37  CPU usage: 72.72% user, 26.81% sys, 0.45% idle
MemRegions: 16780 total, 482M resident, 30M private, 309M shared.
PhysMem: 282M wired, 814M active, 315M inactive, 1412M used, 636M free.
VM: 307G vsize, 1091M framework vsize, 3775808(123) pageins, 1348243(0) pageo
Networks: packets: 37819545/13G in, 41801431/19G out. Disks: 3833135/82G read

PID    COMMAND      %CPU TIME     #TH  #WQ  #POR #MREG RPRVT  RSHRD  RSIZE  V
55822  mdworker     0.0  00:00.15 5    3    55   81    2304K  6920K  6252K  3
55821  screencaptur 0.4  00:00.08 2    1    47   77    576K   6272K  3196K+ 1
55820  main         82.0 00:07.25 1/1  0    17   26    19M+   212K   19M+   3
55797- mdworker32   0.0  00:00.49 3    1    54   92    1904K  15M    5712K  4
```

Insert timers into regions of code

- *Instrumenting* code
- Simple, but incredibly useful
- Runs every time your code is run
- Can trivially see if changes make

```c
struct timeval calc;

tick(&calc);
 /* do work */
calctime = tock(&calc);

printf("Timing summary:\n");
/* other timers.. */
printf("Calc: %8.5f\n", calctime);



void tick(struct timeval *t) {
    gettimeofday(t, NULL);
}


double tock(struct timeval *t) {
    struct timeval now;
    gettimeofday(&now, NULL);
    return (double)(now.tv_sec - t->tv_sec) +
      ((double)(now.tv_usec - t->tv_usec)/1000000.);
}
```

C

SciNet compute • calcul CANADA

Matrix-Vector multiply

- Can get an overview of the time spent easily, because we instrumented our code (~12 lines!)
- I/O huge bottleneck.

```
$ mvm --matsize=2500

Timing summary:
Init:   0.00952 sec
Calc:   0.06638 sec
I/O :   5.07121 sec
```

mat-vec-mult.c

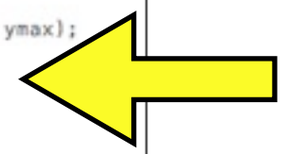
Sampling for Profiling

- How to get finer-grained information about where time is being spent?
- Can't instrument every single line.
- Compilers have tools for *sampling* execution paths.

Program Counter Sampling

- As program executes, every so often (~100ms) a timer goes off, and the current location of execution is recored
- Shows where time is

```
        case SIM_PROJECTILE:
            ymin = xmin = 0.;
            ymax = xmax = 1.;
            dx = (xmax-xmin)/npts;
            dy = (ymax-ymin)/npts;
            init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
            projectile_initvalues(&d, psize, pdens, pvel);
            outputvar = DENSVAR;
            break;
    }

    /* apply boundary conditions and make thermodynamically consistant */
    bcs[0] = xbc; bcs[1] = xbc;
    bcs[2] = ybc; bcs[3] = ybc;
    apply_all_bcs(&d,bcs);
    domain_backward_dp_eos(&d);
    domain_ener_internal_to_tot(&d);


    /* main loop */

    tick(&tt);
    if (output) domain_plot(&d);
    printf("Step\tdt\ttime\n");
    for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

        printf("%d\t%g\t%g\n", step, dt, time);

        if (output && ((step % outevery) == 0) ) {
            sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
            sprintf(binfilename,"dens_test_%d.bin", outnum);
            sprintf(h5filename,"dens_test_%d.h5", outnum);
            sprintf(ncdffilename,"dens_test_%d.nc", outnum);
            domain_output_ppm(&d, outputvar, ppmfilename);
            domain_output_bin(&d, binfilename);
            domain_output_hdf5(&d, h5filename);
            domain_output_netcdf(&d, ncdffilename);
            domain_plot(&d);
            outnum++;
        }
        kl_timestep_xy(&d, bcs, dt);
        apply_all_bcs(&d,bcs);

        kl_timestep_yx(&d, bcs, dt);
        apply_all_bcs(&d,bcs);
    }
    tock(&tt);
```

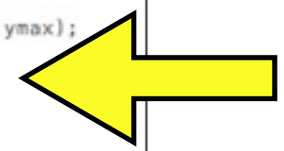


SciNet compute • calcul CANADA

Program Counter Sampling

- Advantages:
  - Very low overhead
  - No extra instrumentation
- Disadvantages:
  - Don't know why code is there

```
        case SIM_PROJECTILE:
            ymin = xmin = 0.;
            ymax = xmax = 1.;
            dx = (xmax-xmin)/npts;
            dy = (ymax-ymin)/npts;
            init_domain(&d, npts, npts, KL_NGUARD, xmin, ymin, xmax, ymax);
            projectile_initvalues(&d, psize, pdens, pvel);
            outputvar = DENSVAR;
            break;
    }

    /* apply boundary conditions and make thermodynamically consistant */
    bcs[0] = xbc; bcs[1] = xbc;
    bcs[2] = ybc; bcs[3] = ybc;
    apply_all_bcs(&d,bcs);
    domain_backward_dp_eos(&d);
    domain_ener_internal_to_tot(&d);


    /* main loop */

    tick(&tt);
    if (output) domain_plot(&d);
    printf("Step\tdt\ttime\n");
    for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {

        printf("%d\t%g\t%g\n", step, dt, time);

        if (output && ((step % outevery) == 0) ) {
            sprintf(ppmfilename,"dens_test_%d.ppm", outnum);
            sprintf(binfilename,"dens_test_%d.bin", outnum);
            sprintf(h5filename,"dens_test_%d.h5", outnum);
            sprintf(ncdffilename,"dens_test_%d.nc", outnum);
            domain_output_ppm(&d, outputvar, ppmfilename);
            domain_output_bin(&d, binfilename);
            domain_output_hdf5(&d, h5filename);
            domain_output_netcdf(&d, ncdffilename);
            domain_plot(&d);
            outnum++;
        }
        kl_timestep_xy(&d, bcs, dt);
        apply_all_bcs(&d,bcs);

        kl_timestep_yx(&d, bcs, dt);
        apply_all_bcs(&d,bcs);
    }
    tock(&tt);
```



gprof for sampling

```
$ gcc -O3 -pg -g mat-vec-mult.c --std=c99
$ icc -O3 -pg -g mat-vec-mult.c -c99
```

turn on profiling

debugging symbols (optional, but more info)

```
$ ./mvm-profile --matsize=2500
[output]
$ ls
Makefile  Mat-vec.dat  gmon.out
mat-vec-mult.c   mvm-profile
```

gprof examines gmon.out

```
$ gprof mvm-profile gmon.out  | more
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.24      0.41     0.41                             main
  0.00      0.41     0.00        3     0.00     0.00  tick
  0.00      0.41     0.00        3     0.00     0.00  tock
  0.00      0.41     0.00        2     0.00     0.00  alloc1d
  0.00      0.41     0.00        2     0.00     0.00  free1d
  0.00      0.41     0.00        1     0.00     0.00  alloc2d
  0.00      0.41     0.00        1     0.00     0.00  free2d
  0.00      0.41     0.00        1     0.00     0.00  get_options
[...]
```

Gives data by function -- usually handy

gprof --line examines gmon.out by line

```
gpc-f103n084-$ gprof --line mvm-profile gmon.out  | more
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 68.46      0.28     0.28                             main (mat-vec-mult.c:82 @ 401
 14.67      0.34     0.06                             main (mat-vec-mult.c:113 @ 40
  7.33      0.37     0.03                             main (mat-vec-mult.c:63 @ 401
  4.89      0.39     0.02                             main (mat-vec-mult.c:112 @ 40
  4.89      0.41     0.02                             main (mat-vec-mult.c:113 @ 40
  0.00      0.41     0.00        3     0.00     0.00  tick (mat-vec-mult.c:159 @ 40
  0.00      0.41     0.00        3     0.00     0.00  tock (mat-vec-mult.c:164 @ 40
  0.00      0.41     0.00        2     0.00     0.00  alloc1d (mat-vec-mult.c:152 @
  0.00      0.41     0.00        2     0.00     0.00  free1d (mat-vec-mult.c:171 @
  0.00      0.41     0.00        1     0.00     0.00  alloc2d (mat-vec-mult.c:130 @
  0.00      0.41     0.00        1     0.00     0.00  free2d (mat-vec-mult.c:144 @
  0.00      0.41     0.00        1     0.00     0.00  get_options (mat-vec-mult.c:1
400a30)
```

gprof hands-on

- Edit your makefile to include -g -pg lines, rebuild your code
- Where does your code spend most of its time -- by function? By line?

SciNet compute • calcul CANADA

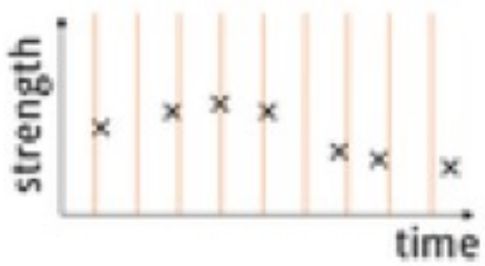
Object Oriented Python

- Interpolation
- Example from http://software-carpentry.org/4_0/oop

SciNet
compute • calcul
CANADA

1d Interpolation

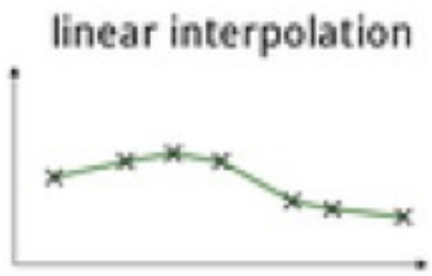
- Take discrete, irregular timeseries
- Allow sampling at any time
- For integration, averaging, etc.



SciNet compute • calcul CANADA

1d Interpolation

- Many ways to do this
- Piecewise constant (step function)
- Linear interpolation, etc.



SciNet compute • calcul CANADA

Obj. Oriented Python

- Similar to C++
- constructor: __init__.
- Methods are just functions ‘in’ the class
- Local values: self.values

```
class StepSignal(Object):

   def __init__(self,values):
      '''Values is ((x0,y0),(x1,y1)...)'''
      self.values = values

def get(self,where):
   assert where >= self.values[0][0]
   for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
         return y0
   assert where < x1
```

Obj. Oriented Python

- Initialize with the data
- call .get method to sample at given point

```
class StepSignal(Object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def get(self,where):
        if where < self.values[0][0]:
            raise IndexError, '%f too low' % where
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return y0
        raise IndexError, '%f too high' % where
```

```
>>> import stepsignal
>>> interp = stepsignal.StepSignal(((0.,0.),
              (1.,1.), (2.,2.)))

>>> for x in (0.0, 0.5, 1.1, 1.75):
...     print x, interp.get(x)
...
0.0 0.0
0.5 0.0
1.1 1.0
1.75 1.0
```

SciNet compute • calcul CANADA

Obj. Oriented Python

- Error handling works as expected
- This would be a reasonable set of tests for this class

```
class StepSignal(object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def get(self,where):
        assert where >= self.values[0][0]
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return y0
        assert where < x1
```

```
>>> interp.get(-.001)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "stepsignal.py", line 8, in get
    assert where >= self.values[0][0]
AssertionError

>>> interp.get(2.1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "stepsignal.py", line 14, in get
    assert where < x1
AssertionError
```

SciNet compute • calcul CANADA

Linear Interpolation

- Different class, with different interpolation
- Test this:

```
class LinearSignal(object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def get(self,where):
        assert where >= self.values[0][0]
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return y0 + (y1-y0)*(where-x0)/(x1-x0)
        assert where < x1
```

Linear Interpolation

- Different class, with different interpolation
- Test this:

```
class LinearSignal(object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def get(self,where):
        assert where >= self.values[0][0]
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return y0 + (y1-y0)*(where-x0)/(x1-x0)
        assert where < x1
```

```
>>> import linearsignal
>>> interp = linearsignal.LinearSignal(((0.,0.),
            (1.,1.), (2.,2.)))

>>> for x in (0.0, 0.5, 1.1, 1.75):
...     print x, interp.get(x)
...
0.0 0.0
0.5 0.5
1.1 1.1
1.75 1.75
```

SciNet compute • calcul CANADA

Using the class

- Argument "signal" can be anything with a get() method
- Can pass it linear signal or step signal, or..

```
def integrate(signal, x0, x1, num_samples):
    width = (x1 - x0)/num_samples
    total = 0.0
    for i in range(num_samples):
        x = x0 + i*width
        total += signal.get(x)*width
    return total
```

Using the class

- Argument "signal" can be anything with a get() method
- Can pass it linear signal or step signal, or..

```
class Sinusoid(object):

    def __init__(self, amplitude, frequency):
        self.amp = amplitude
        self.freq = frequency

    def get(self, x):
        return self.amp * math.sin(x * self.freq)
```

Inheritance

- Two classes are almost identical
- Want to reduce code overhead
- Also, make explicit that they have same interface

```
class StepSignal(object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def get(self,where):
        assert where >= self.values[0][0]
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return y0
        assert where < x1
```

```
class LinearSignal(object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def get(self,where):
        assert where >= self.values[0][0]
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return y0 + (y1-y0)*(where-x0)/(x
        assert where < x1
```

SciNet compute • calcul CANADA

Inheritance

- Find common code
- Initialization
- Find routine (which get will be based on)
- Not very useful in and of itself

```
class InterpolatedSignal(object):

    def __init__(self,values):
        '''Values is ((x0,y0),(x1,y1)...)'''
        self.values = values

    def find(self,where):
        assert where >= self.values[0][0]
        for i in range(len(self.values)-1):
            x0, y0 = self.values[i]
            x1, y1 = self.values[i+1]
            if x0 <= where <= x1:
                return i
        assert where < x1

    def get(self, where):
        raise NotImplementedError('Must provide get
```

Inheritance

- Define implementations that inherit from Interpolated Signal
- Just the code that's different
- Note get is overridden

```
class StepSignal(InterpolatedSignal):

    def get(self, where):
        i = self.find(where)
        return self.values[i][0]


class LinearSignal(InterpolatedSignal):

    def get(self, where):
        i = self.find(where)
        x0 = self.values[i][0]
        x1 = self.values[i+1][0]
        y0 = self.values[i][1]
        y1 = self.values[i+1][1]
        return y0 + (y1-y0)/(x1-x0)*(where-x0)
```

```
>>> import interpolatedsignal
>>> interp =
interpolatedsignal.StepSignal(((0.,0.),(1.,1.),
(2.,2.)))
>>> interp.get(1.5)
1.0
```

SciNet compute • calcul CANADA

Inheritance

- Can put better input validation into the base class
- Make sure values are pairs, increasing in x...
- Both subclasses get those improvements automatically

```
class StepSignal(InterpolatedSignal):

    def get(self, where):
        i = self.find(where)
        return self.values[i][0]


class LinearSignal(InterpolatedSignal):

    def get(self, where):
        i = self.find(where)
        x0 = self.values[i][0]
        x1 = self.values[i+1][0]
        y0 = self.values[i][1]
        y1 = self.values[i+1][1]
        return y0 + (y1-y0)/(x1-x0)*(where-x0)
```

```
>>> import interpolatedsignal
>>> interp =
interpolatedsignal.StepSignal(((0.,0.),(1.,1.),
(2.,2.)))
>>> interp.get(1.5)
1.0
```

SciNet compute • calcul CANADA