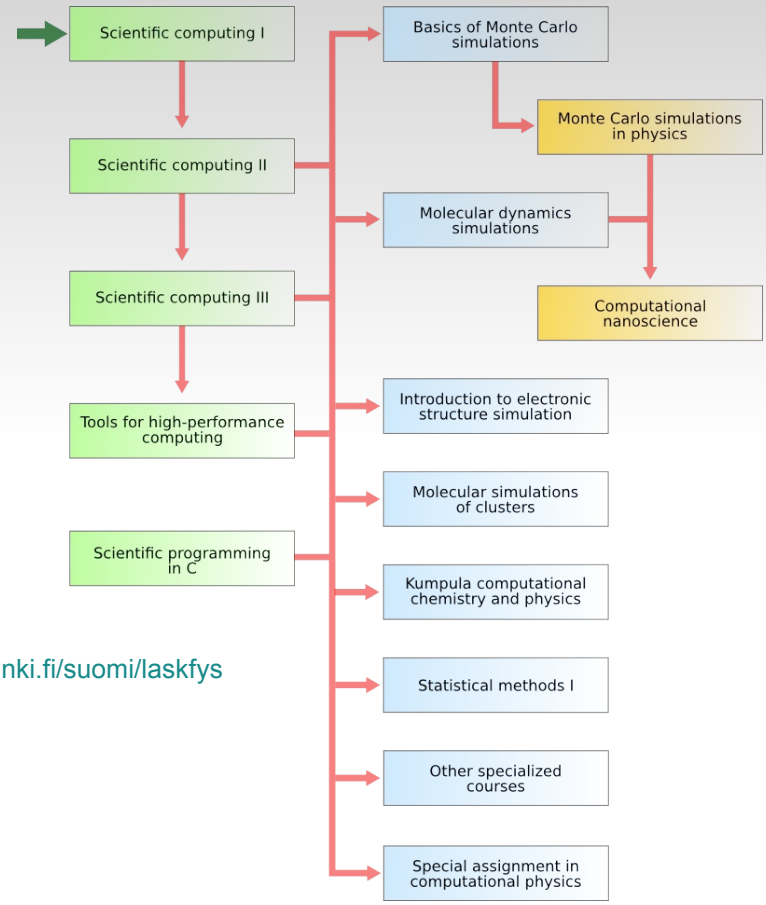
53369
Scientific computing III
Tieteellinen laskenta III

- **Lectures**: Mon 12-14 ,Wed 12-14, room B121 (Exactum)
- **Excercises**: TBA
- **Lecturer**: Antti Kuronen, antti.kuronen@helsinki.fi
- **Assistants**: Ane Lasa (ane.lasa@helsinki.fi), Andrey Ilinov (andrey.ilinov@helsinki.fi)
- **Course homepage**: http://www.physics.helsinki.fi/courses/s/tl3/

- **Objectives**:
  - To familiarize oneself with the most common numerical methods and algorithms
  - To learn to apply these algorithms by using
    - self-made programs
    - numerical libraries
    - numerical programs.

Specialization alternative in computational physics



http:///www.physics.helsinki.fi/suomi/laskfys

Course material

- These lecture notes on course home page
  http://www.physics.helsinki.fi/courses/s/tl3/
  - Links to useful material online.
  - List of books on numerical methods
- Example programs presented in lecture notes can be downloaded from http://www.physics.helsinki.fi/courses/s/tl3/progs/

Exercises

- Return by email to either ane.lasa@helsinki.fi or andrey.ilinov@helsinki.fi
- Details of the returning times will be given later
- Small programming tasks
  - Return (as a tar or zip archive)
    - Program code
      - With possible compilation instructions or Makefile
        - Must compile without errors.
      - Input and output files
      - Text that explains what you have done
        - Plain ASCII or (even better) PDF
        - No Word or LaTeX documents
      - Plots and figures!
    - Detailed instructions on naming your files can be found at http://www.physics.helsinki.fi/courses/s/tl3/exercises/exercisefiles.pdf

Table of contents

1) Introduction: tools, computing environment in Kumpula, visualization
2) Basic things in numerics: floating point numbers, error sources
3) Linear algebra: equations, decompositions
4) Eigenvalue problems
5) Nonlinear equations: bisection, secant, Newton
6) Interpolation: polynomes, splines, Bezier curves
7) Numerical integration: trapeziodal, Romberg, Gauss
8) Function minimization: Newton, conjugate gradient,stochastic methods
9) Generation of random numbers: LCG, shift register, non-uniform RNs
10) Statistical description of data: prob. distributions, comparison of data sets
11) Modeling of data: linear and nonlinear fitting
12) Fourier and wavelet transformations: FFT, DWT, applications
13) Differential equations: ODE's, PDE's

Prerequisites

1) Mathematics on the level of MAPU (Mathematics for physicists)
2) Programming using C or Fortran90/95/2003 languages
   - Programming is not taught in this course.

- In addition, you need access to University Linux system (punk.it.helsinki.fi, mutteri.it.helsinki.fi)...
- ... or to some other Linux/Unix system that has the compilers, libraries and matlab installed.
  - Linux (or Mac) machine at home is fine for self-made programs.

- Of course, you may work in Windows environment if you like and if you have all the tools you need.

Tools

- Self-made (or colleague-made) programs (+ numerical libraries)
  - In many cases the most efficient solution.
  - Particularly, when used with highly optimized numerical libraries (don't reinvent the wheel).
- Numerical software (in practice **`matlab`** and its clones **`octave`**, **`scilab`**)
  - A perfect fit if your problem can be cast in a standard form (e.g. standard linear algebra operations) or if you need not do the calculation many times.
  - In other cases not necessarily the most efficient solution.
  - **`matlab`**: installed on punk.it.helsinki.fi and mutteri.it.helsinki.fi
- Symbolic mathematical software (**`mathematica`**, **`maple`**, **`maxima`**)
  - For non-numerical calculations.
  - Output in programming languages: results used in ones own codes.
  - **`maple`**: installed on punk.it.helsinki.fi and mutteri.it.helsinki.fi
  - **`maxima`**: open source software

Tools

- Example (from my own work):
  - Field: computational materials and medical physics
  - Tools almost entirely self-made codes (or codes obtained from colleagues)
  - Commercial codes: *ab initio* electronic structure calculations
  - Numerical libraries used (linear algebra, special integrals, integration...)
  - Data-analysis also done mostly by self-made codes (and **`gawk`** one-liners); sometimes **`matlab`** (all eigenvalues of a large matrix)
  - Symbolic calculations: calculate e.g. the gradient of a complex many-body potential; check partial results by **`maple`**, or **`maxima`**
  - Visualization by molecular visualization codes; not self-made but free (open source), 3D plots by **`matlab`**

Tools: self-made codes

- Either C (C++) or Fortran90
  - Which one to choose?
  - The following is a bit personal opinion.
    - C
      + Common: compilers found on every architecture.
      - Low level of abstraction (arrays==pointers etc.).
    - Fortran90
      + High level of abstraction.
      - Compiler not available on every system. (But gfortran!)
  - Both languages allow you to write ugly code!
  - Not necessarily essential differences in speed of execution.
  - If you know one language the other one is not hard to learn.
  - And remember: Fortran is not dead! F2008 is here.

Tools: self-made codes

- **General programming issues**
  - Think before you start
    - Make a block diagram of the program.
    - Think about subroutine interfaces (what parameters in and out).
    - Write out explicitely subroutine interfaces (prototypes, interface modules).
    - Modularize your code
    - Divide the code into smaller parts:
      - Define interfaces.
      - Code and test the parts separately.
    - Generalize slightly
      - One day you will need to use your code in a slightly different problem.
      - Or the size of the problem changes
      - Foresee this by e.g. not fixing array sizes.
    - Make the code readable
      - Use meaningful variable and function names.
      - Indent code (use language-aware editor; e.g. **`emacs`**, **`gedit`**, **`vim`** ...)
    - Think about efficiency.
      - Concentrate on the most time-consuming parts of the code (profiling).
      - Optimize also your own time.

Tools: self-made codes

- **General programming issues** (continued)
  - Program input
    - Large amounts of data from files
    - Often changing parameters from command line

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main (int argc, char **argv)
{
  double x,y;
  int i;

  if (argc!=4) {
    fprintf(stderr,"Usage: %s x y i\n",argv[0]);
    return (1);
  }

  x=atof(*++argv);
  y=atof(*++argv);
  i=atoi(*++argv);
  fprintf(stdout,"x %g y %g i %d\n",x,y,i);

  return(0);
}
```

```
program argtest
  implicit none
  character (len=80) :: argu
  integer :: command_argument_count,i
  real :: x,y

  call get_command_argument(0,argu)
  if (command_argument_count()/=3) then
     call get_command_argument(0,argu)
     write(0,'(a,a,a)') 'Usage: ',trim(argu),' x y i'
     stop
  endif

  call get_command_argument(1,argu) ;read(argu,*) x
  call get_command_argument(2,argu) ;read(argu,*) y
  call get_command_argument(3,argu) ;read(argu,*) i
  write(6,'(a,g10.4,a,g10.4,a,i5)') 'x ',x,' y ',y,' i ',i

  stop
end program argtest
```

Note that the routines `command_argument_count` and `get_command_argument` are part of the Fortran 2003 standard. Most compilers support also the de facto standard routines `iargc` and `getarg`:

`command_argument_count()` → `iargc()`
`get_command_argument(i,argu)` → `getarg(i,argu)`

Tools: Linux environment

- **Linux environment in general**
  - Personal opinion: the best environment for scientific computation

    - Fortran90/95/2003 compilers
      - Many commercial (Absoft, Portland Group, Pathscale, ...)
      - Intel Fortran compiler for Linux: free for personal use
        http://software.intel.com/en-us/non-commercial-software-development
        - Generates fast code even for AMD processors
      - GNU Fortran compiler (**`gfortran`**)
        - A member of the GNU compiler collection (http://gcc.gnu.org/wiki/GFortran)
      - Open64 (http://www.open64.net/)

    - C compilers
      - GNU C compiler (**`gcc`**) comes with every Linux distribution
        - Can also be installed on Windows systems
      - Intel C compiler for Linux: free for personal use
        http://software.intel.com/en-us/non-commercial-software-development
        - Generates faster code than **`gcc`**
      - Open64 (http://www.open64.net/)

Tools: Linux environment

- **University Linux programming environment**
  - Machines punk.it.helsinki.fi and mutteri.it.helsinki.fi
    - Fortran90 compiler **`gfortran`**:

```
> gfortran -o hello hello.f90
> ./hello
Hello world!
```

```
program hello
  write(6,*) 'Hello world!'
  stop
end program hello
```

    - C compiler **`gcc`**:

```
> cc -o hello hello.c
> ./hello
Hello world!
```

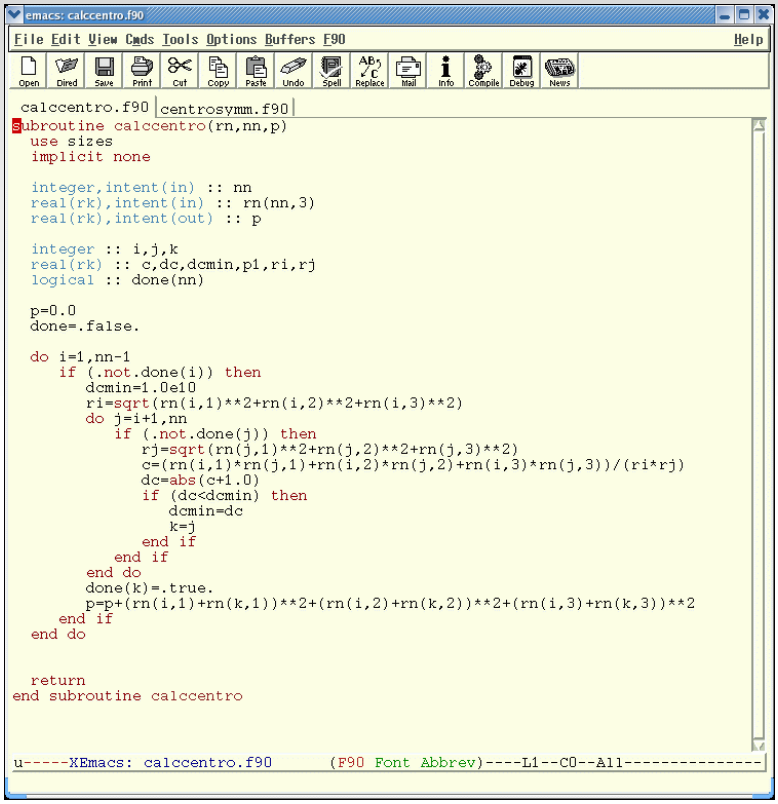
```
#include <stdio.h>
int main()
{
  printf("Hello world!\n");
  return 0;
}
```

  - The most important options (see **`man gcc`** and **`man gfortran`**)

`-o`	specify output file
`-c`	compile only
`-O<n>`	optimization level (<n>=0..5)
`-g`	include debugger info to code
`-L<dir>`	search libraries from <dir>
`-l<lib>`	include library <lib>
`-fbounds-check`	check array bounds (only for `gfortran`)

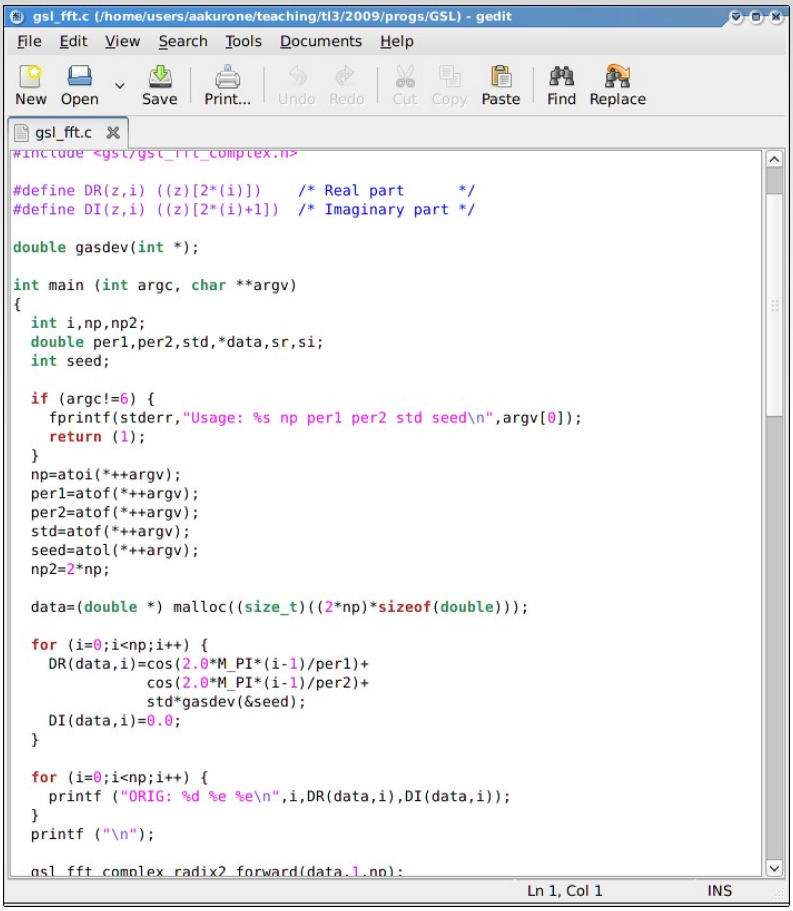
Tools: Linux environment

- **Editors**
  - **GNU Emacs, XEmacs**
    - syntax hilightning for many languages
    - automatic indentation
    - abbreviation mode
    - compilation and debugging
    - included in every Linux distribution
    - available also for most other Unix platforms and for Windows (http://www.cygwin.com/)
    - extensive documentation
    - no limit for customization
    - includes also a psychotherapist (command `M-x doctor`)



Tools: Linux environment

- **Editors** (continued)
  - Linux distributions include many other editors suitable for program development
    gedit
    kate
    vim



Tools: Linux environment

- **Debuggers**
  - Linux: GNU debugger gdb
    - compile code with: `gcc -g -O0`
    - example

```
gdb mdmorse
GNU gdb Red Hat Linux (6.1post-1.20040607.43rh)
...
(gdb) b GetForces
Breakpoint 1 at 0x804aaca: file forces.c, line 19.
(gdb) run
Starting program: /home/users/akuronen/c/mdmorse
...
Breakpoint 1, GetForces (x=0xbfede670, a=0xbfe69330, N=500, size=
      {x = 18.126900793000001, y = 18.126900793000001, z = 18.126900793000001}, periodic=
      {x = 1, y = 1, z = 1}, m=63.545999999999999, neighbourlist=0xbfa655a0, rpotcut=5,
    Epot=0xbfe55aa0, virial=0xbfa654b8) at forces.c:19
19        half.x=size.x/2.0;
(gdb) list
14      {
15        struct vector   half;
16        int        i,j,jj,startofineighbourlist,nneighboursi;
17        double   r,rsq,rpotcutsq,dx,dy,dz,help1,help2,V,dVdr;
18
19        half.x=size.x/2.0;
20        half.y=size.y/2.0;
...
```

set breakpoint

stopped after calling GetForces

list source near breakpoint

    - Technical student's debugger (teekkarin debuggeri): `fprintf(stderr,...);`

More on debugging, profiling, make system on course **Tools of High Performance Computing**. http://www.physics.helsinki.fi/courses/s/stltk/ Next time given in autumn 2013.

Tools: Linux environment

- **Compiling large programs: `make`**
  - Large program: better to split into many files
    - No need to compile all the files all the time: use command `make`!
    - File called `Makefile` or `makefile` contains all nterfile dependencies
    - When one file chances all those files that depend on it have to be update.
    - Just give command make in the directory where the Makefile is located
    - More info: `man make`

variable definitions

```
CC= cc -g -O0
HEADERS=global.h
OBJECTS= main.o inout.o physical.o neighbourlist.o solve.o forces.o
TARGET= mdmorse
mdmorse:                $(OBJECTS)
                        $(CC) -o $(TARGET) $(OBJECTS)  -lm
```

dependencies

```
main.o:                 main.c $(HEADERS)
                        $(CC) -c main.c
inout.o:                inout.c $(HEADERS)
                        $(CC) -c inout.c
physical.o:             physical.c $(HEADERS)
                        $(CC) -c physical.c
neighbourlist.o:        neighbourlist.c $(HEADERS)
                        $(CC) -c neighbourlist.c
solve.o:                solve.c $(HEADERS)
                        $(CC) -c solve.c
forces.o:               forces.c $(HEADERS)
                        $(CC) -c forces.c
clean:
                        rm -f *.o $(TARGET)
```


Tools: Linux environment

- **Compilation: libraries**
  - When compiling you need to tell
    - Which libraries to use:

```
f90 -o nagtest nagtest.f90 -lnag
cc -o nagtest nagtest.c -lnag -lfor -lm
```

      - Option `-lfile` looks for library file libfile.so or libfile.a from predetermined directories (normally at least `/usr/lib` and `/lib`)
    - Where to find them.
      - Option `-L` tells to first look for library files in particular directories. Example:
        `cc -o prog prog.c -L/home/user/libs/ -lmylib`

Tools: Linux environment

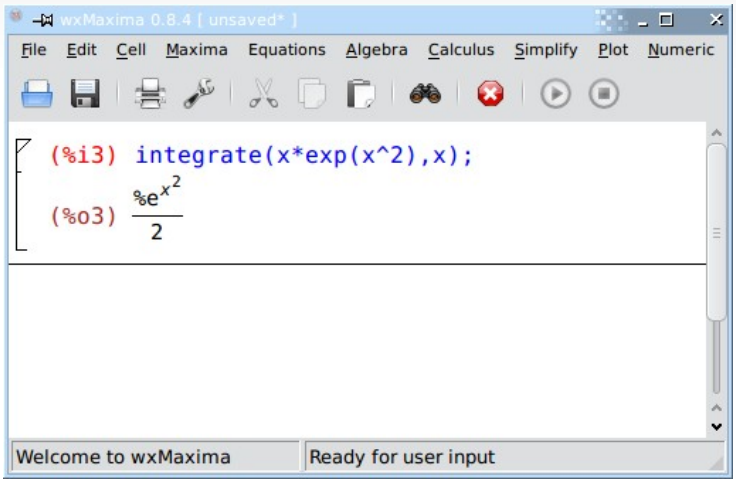
- **Numerical libraries**
  - **NAG**
    - Extensive commercial library
    - Installed on mutteri.it.helsinki.fi
  - **LAPACK**
    - De facto standard in linear algebra
    - Open source code
    - Architecture optimized implementations available (commercial)
    - http://www.netlib.org/lapack/
    - Uses the BLAS library (Basic Linear Algebra Subroutines; http://www.netlib.org/blas/
  - **GNU Scientific Library (GSL)**
    - Open source numerical library for C and C++
    - http://www.gnu.org/software/gsl/
    - Includes a wide range of numerical routines:
      Complex Numbers, Roots of Polynomials, Special Functions, Vectors and Matrices, Permutations, Sorting, BLAS Support, Linear Algebra ,Eigensystems, Fast Fourier Transforms, Quadrature, Random Numbers, Quasi-Random Sequences, Random Distributions, Statistics, Histograms, N-Tuples, Monte Carlo Integration, Simulated Annealing, Differential Equations, Interpolation, Numerical Differentiation, Chebyshev Approximation, Series Acceleration, Discrete Hankel Transforms, Root-Finding, Minimization, Least-Squares Fitting, Physical Constants, IEEE Floating-Point

Tools: Linux environment

- **Numerical libraries** (continued)
  - **SLATEC**
    - Open source: http://www.netlib.org/slatec/
    - A comprehensive software library containing over 1400 general purpose mathematical and statistical routines.
    - Written in Fortran 77.
    - A Fortran 90 version can be found at https://people.scs.fsu.edu/~burkardt/f_src/slatec/slatec.html

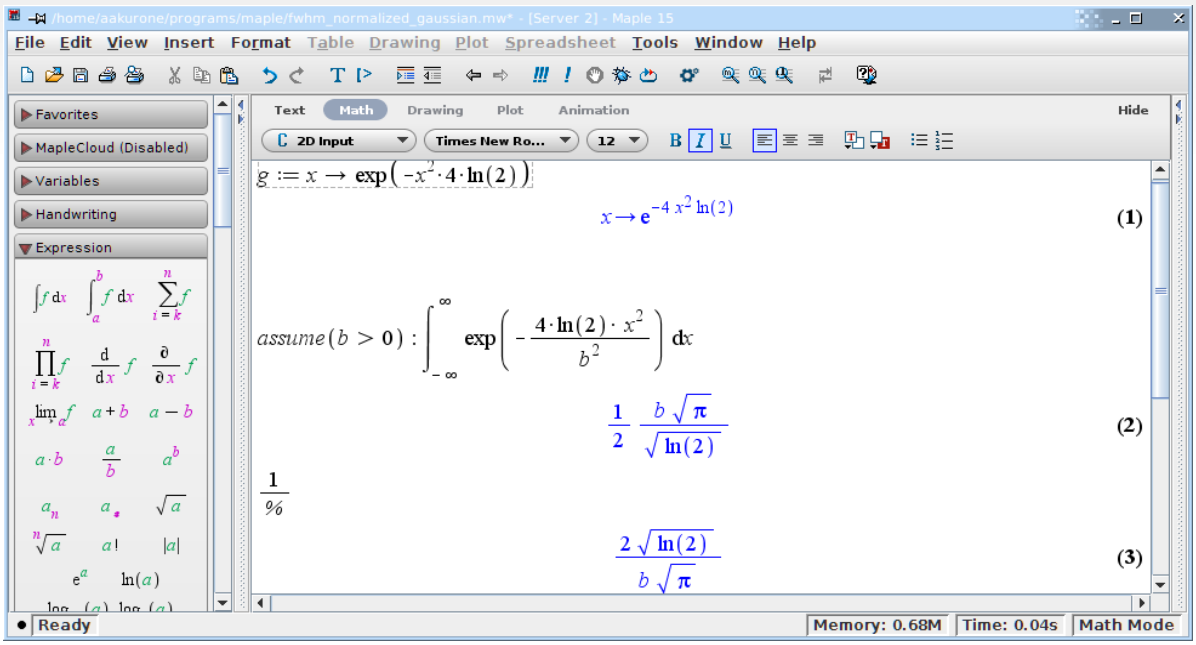
Tools

- **Symbolic computing**
  - **`maxima`**: an open source symbolic computation system based on macsyma
    - http://maxima.sourceforge.net/
    - Ported to linux and Windows
    - **`wxmaxima`** is a user-friendly interface to **`maxima`**



Tools

- **Symbolic computing** (continued)
  - **`maple`**: a commercial symbolic computation system
    - Nice and quite usable graphical interface.
    - Installed on mutteri.it.helsinki.fi and on the machines in the computer class



Tools

- **CPU time counting routines**
  - C: `clock()/CLOCKS_PER_SEC` CPU time used in seconds
  - Fortran90: routine `CPU_TIME()`

```
REAL time_begin, time_end
    ...
CALL CPU_TIME(time_begin)
    ...
CALL CPU_TIME(time_end)
PRINT (*,*) 'CPU time ', time_end-time_begin, ' seconds'
```

- **Profiling**
  - Compile with -pg option
    `f90 -pg -o prog prog.f90`
  - Run
    `./prog`
  - Generate statistics
    `gprof prog gmon.out`

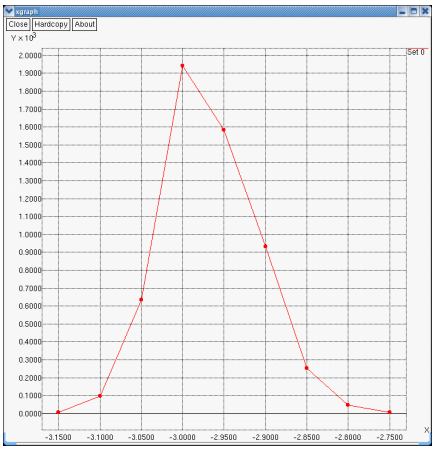
```
Flat profile:

Each sample counts as 0.000976562 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 48.36      0.25     0.25  1000000     0.25     0.25  gauss2_
 24.47      0.37     0.12        1 124023.44 369140.62  teht8_
 11.75      0.43     0.06                             log
  9.83      0.48     0.05                             floorf
  2.31      0.49     0.01                             sqrt
  1.35      0.50     0.01                             write
  0.39      0.50     0.00                             cvt_ieee_s_to_text_ex
  0.39      0.50     0.00                             cvtas_s_to_a
  0.39      0.50     0.00                             for__format_value
  0.19      0.50     0.00                             cvt_integer_to_text
  0.19      0.50     0.00                             for__desc_ret_item
  0.19      0.51     0.00                             for_write_seq_lis
  0.19      0.51     0.00                             for_write_seq_lis_xmit
 .
```


Tools

- **Unix/Linux has many small but handy tools**
- Swiss army knife of Unix: awk (or gawk)
  - A lot can be done using simple 'one-liners'
  - Example: we have a data file:
  - We want to plot the distribution of the 5th column data of those lines that have 'Cu' as the 1st string on the line

```
         500
 mdmorse atom output at time       2.000 fs
 Cu    -8.151144   -8.155904   -8.149658   -3.033020
 Cu    -6.336929   -6.348670   -8.163063   -3.040200
 Cu    -8.152819   -6.338272   -6.340627   -3.050315
 Cu    -6.341799   -8.167974   -6.346826   -3.033914
 Cu    -8.158021   -8.155310   -4.520418   -3.051445
 Cu    -6.351106   -6.347753   -4.540661   -3.038614
 Cu    -8.156158   -6.336749   -2.716651   -3.043019
 Cu    -6.343872   -8.161483   -2.726575   -3.028146
```

```
gawk 'BEGIN {c=20} $1=="Cu" {i=int(c*$6+0.5); e[i]++}
END {for (i in e) print i/c,e[i]}' atoms.out | sort -n
| xgraph -P
```



- Quick and dirty plotting: `xgraph`
- Remember also `perl`
- These tools reduce the need to build C or F90 programs or to launch Matlab for every small task.

Tools

- **gawk** command line syntax
  ```
  gawk -f progfile [--] file …
  gawk 'program' file …
  ```
- **gawk** program
  ```
  pattern {action_statements}
  ```

Pattern:
```
BEGIN
END
/regular expression/
relational expression
pattern && pattern
pattern || pattern
pattern ? pattern : pattern
(pattern)
! pattern
pattern1, pattern2
```

  - Read the input file and when pattern is found execute the `action_statements`.
  - C-like syntax
  - Variables: no need to declare, interpretation depends on the context.
  - Good string handling functions, mathematical functions (incl. trigonometry)
  - Special variable: `NR` line number of current file, `NF` number of fields in the current line, `$i` ith field in the current line (fields separated by spaces, but that can be changed).
  - Simple example: print the 4th field of every line that has the string 'dat' in the very beginning of the line:
    ```
    gawk '/^dat/ {print $4}' file.dat
    ```
  - For more information: `man gawk`, `man awk`.

Tools

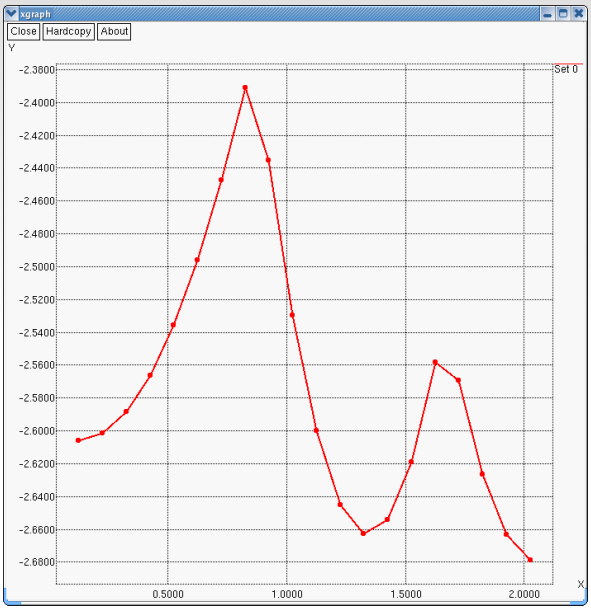
- Other tools
  ```
  sort
  uniq
  sed
  tr
  …
  ```
- Use the `man` command to find more information.

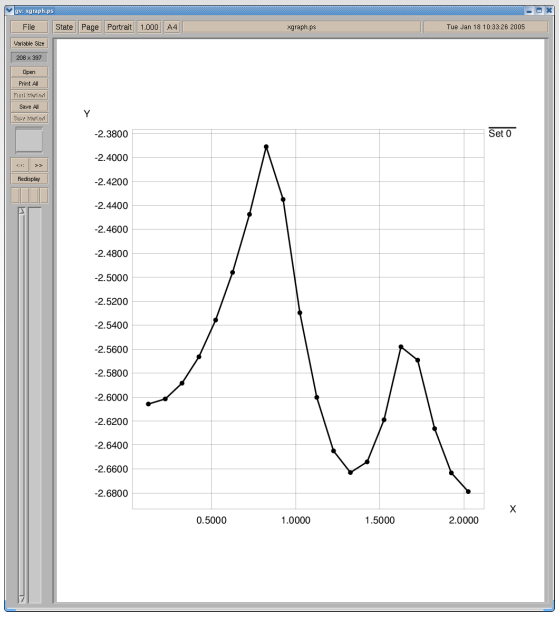
Tools

- **Visualization**: not only fun
- In many cases essential in order to understand the results of computations.
- Simple 2D visualization (curves): e.g. abovementioned `xgraph`
  - Fast, run entirely from command line, good for everyday use
    - Not publication quality figures
    - Reads data in the form of x,y pairs from a file (or from stdin)
    - Open source; package for many Linux distributions.

Tools

- **Visualization** (continued)
  - Dump window to a Postscript file.





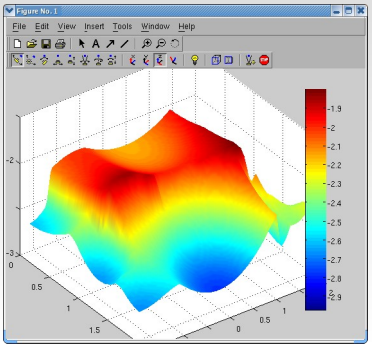
- See also **gnuplot** (http://www.gnuplot.info/) and **xmgrace** (http://plasma-gate.weizmann.ac.il/Grace/)

Tools

- **3D visualization** (surfaces, contours, vector fields):

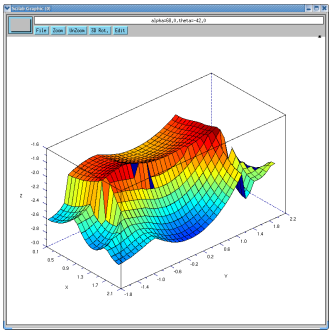
matlab

```
dx=(x2-x1)/npoints;
dy=(y2-y1)/npoints;
[xi,yi]=meshgrid(x1:dx:x2,y1:dy:y2);
zi=griddata(x,y,z,xi,yi,'linear');
surf(xi,yi,zi)
shading interp
axis([x1 x2 y1 y2])
colorbar
```



scilab

```
x=read("x.dat",-1,1);
y=read("y.dat",-1,1);
z=read("z.dat",-1,39);
plot3d(x,y,z);
f=get("current_figure");
f.color_map=jetcolormap(128);
h=get("hdl");
h.color_flag=1;
```

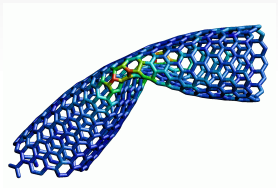


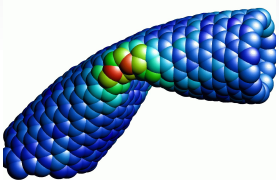
http://www.scilab.org/

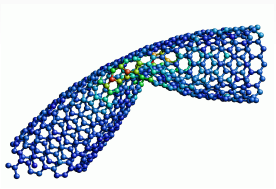
paraview: http://paraview.org

Tools

- **3D visualization:** Molecular visualization
  - Many good open source programs
    - **JMol** (http://jmol.sourceforge.net/)
    - **OVITO** (http://www.ovito.org/)
    - **RasMol** (http://www.umass.edu/microbio/rasmol/)
      - A bit old; can not utilize modern GPU's properly



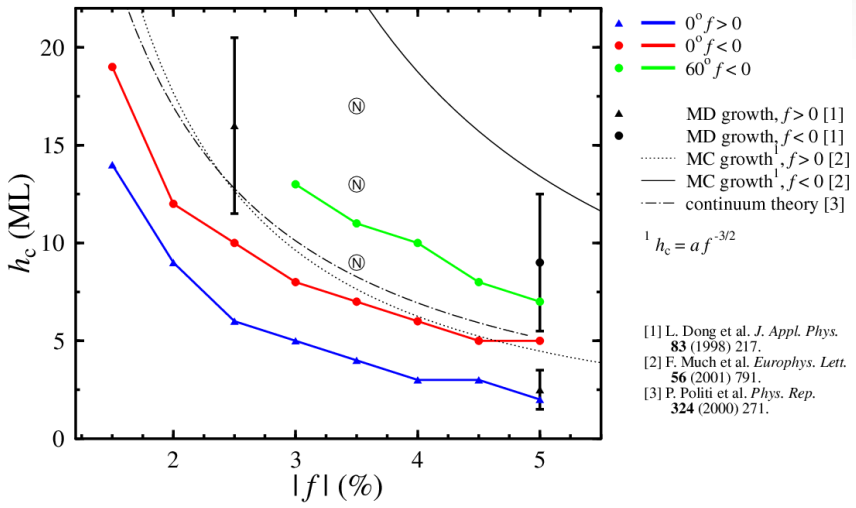




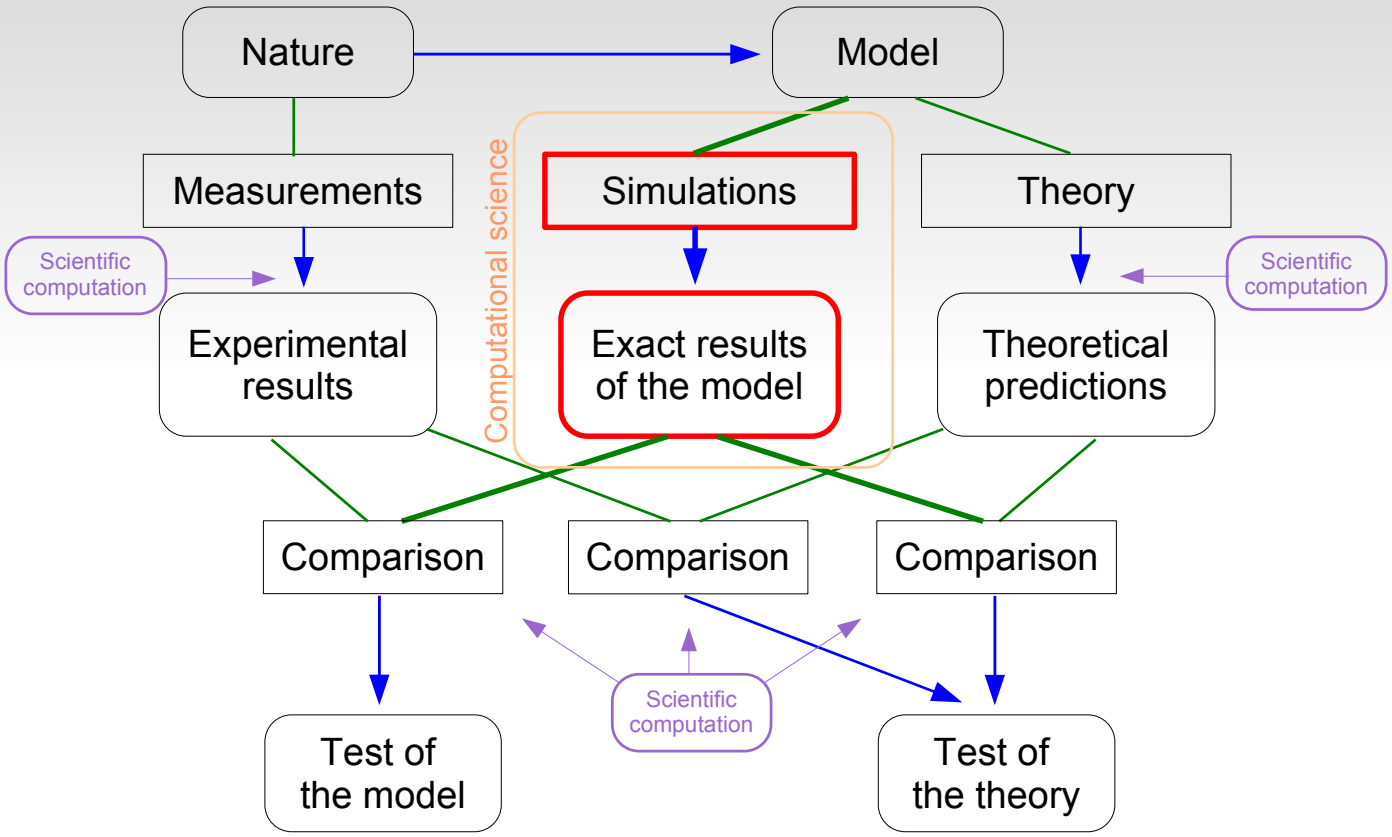
Tools

- **Publication quality figures**
  - **`Matlab`**
  - Open source programs:
    - **`gle`** (http://glx.sourceforge.net/)
    - **`xmgrace`** (http://plasma-gate.weizmann.ac.il/Grace/)

Example by GLE:



Computational science and scientific computation



Computational science and scientific computation

- Simulations:
  - An algorithmic approach; no shortcut to final results.
  - Often the only way to obtain results of a model.

- Computational science
  - Often seen as a method.
  - Some people claim that it is becoming a field of science of its own?