

Laboratory of computational physics part 1: classical physics

Franco Bagnoli

Topics of the course (part 1):

Integration of differential equations (RK and Verlet).

Application to the harmonic oscillator and the pendulum.

Molecular dynamics of a Lennard-Jones gas. Measurement of physical quantities.

Bifurcations and chaos: the logistic map. Lorenz's system.

Lyapunov exponents in maps and continuous systems.

Attractors

Stochastic systems: random walk. Probability distribution.

Diffusion.

Percolation: Markov processes. Connection with the mean field.

Monte Carlo method. The Ising model. Phase transitions, fluctuations.

Disordered systems, spin glasses, neural networks (outline).

Tools

C+gnuplot

Matlab or Octave

elements of shell (bash), unix file system, common unix commands.

Prerequisites:

algebra (vector, matrices)

calculus (functions, derivatives, partial derivatives, gradient, etc.)

basic knowledge of the C language

Editor

nano (minimal editor); vi (powerful but complex)

emacs (powerful but weird)

mc (if installed: file manager and editor)

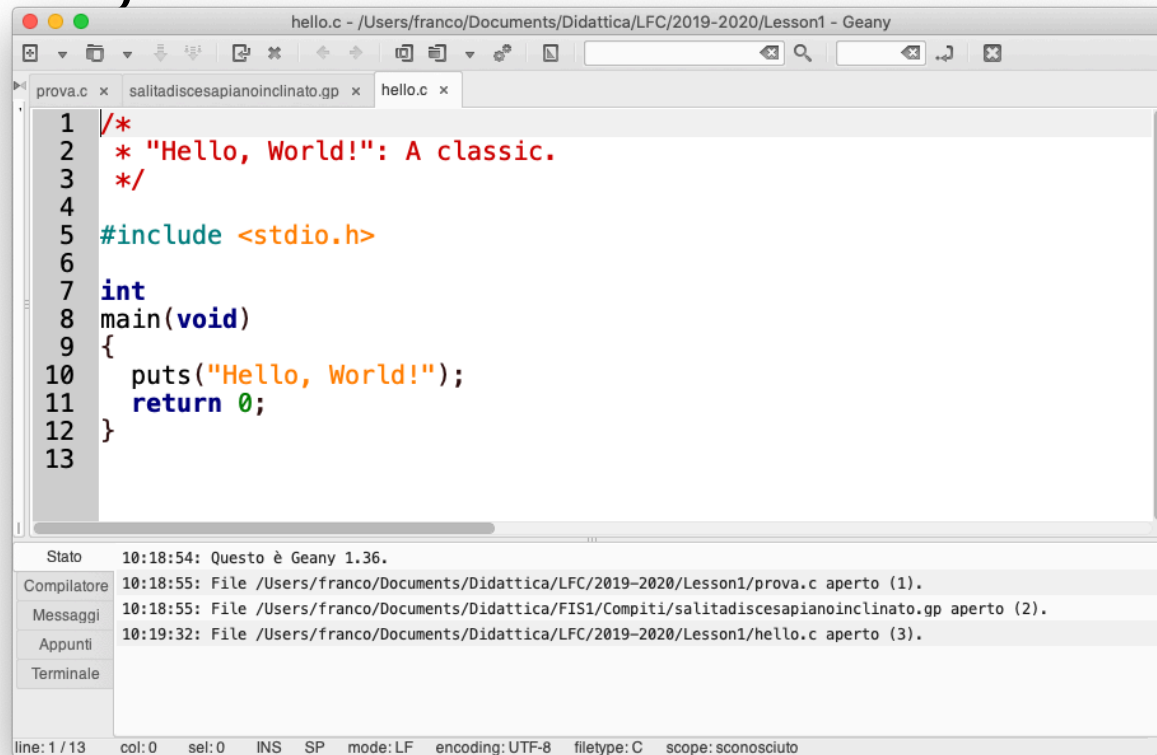
ne (nice editor)

or geany

(also on

windows,

macosx)



The screenshot shows the Geany text editor window titled "hello.c - /Users/franco/Documents/Didattica/LFC/2019-2020/Lesson1 - Geany". The editor contains the following C code:

```
1  /*
2  * "Hello, World!": A classic.
3  */
4
5  #include <stdio.h>
6
7  int
8  main(void)
9  {
10     puts("Hello, World!");
11     return 0;
12 }
13
```

Below the editor, the status bar shows "Stato 10:18:54: Questo è Geany 1.36." and a log of recent actions:

- Compilatore 10:18:55: File /Users/franco/Documents/Didattica/LFC/2019-2020/Lesson1/prova.c aperto (1).
- Messaggi 10:18:55: File /Users/franco/Documents/Didattica/FIS1/Compiti/salitadiscsapianoinclinato.gp aperto (2).
- Appunti 10:19:32: File /Users/franco/Documents/Didattica/LFC/2019-2020/Lesson1/hello.c aperto (3).
- Terminale

The status bar at the bottom indicates "line: 1 / 13 col: 0 sel: 0 INS SP mode: LF encoding: UTF-8 filetype: C scope: sconosciuto".

Installation

Linux: use the package manager to install gnuplot-x11, gcc (probably already present) and geany (or `code::blocks`)

MacOSX: install Apple's [Xcode](#) Developer Tools and macports (or homebrew), then from the package manager install gnuplot-wx or gnuplot-X11 (port install ...) and geany

Windows: install Cygwin and from the package manager install xinit, gnuplot, gcc-g++ and geany (or `code::blocks`)

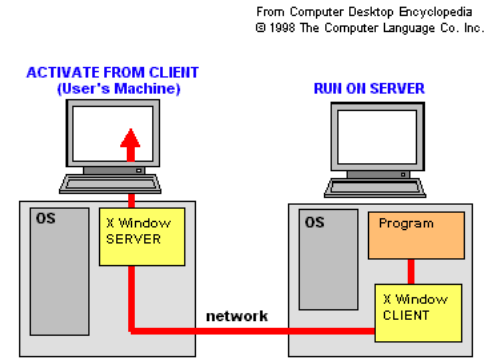
Why all this stuff?

Unix (linux) separates where programs run from where the output is sent, through a protocol (X11) that runs on Internet. So it is quite easy to visualize the output on another pc, but one has to understand the serve/client architecture.

The pc with the screen is called the X11 server, and the pc that runs the program (your terminal) is the client. So you have to run the server program (in linux is the default, in MacOSX it is called Xquartz, on windows it is started with "startxwin") and tell the client where the server is.

So the first step is

`startxwin &` (so that it goes in background)



Why all this stuff?

Then one has to specify the DISPLAY variable

```
export DISPLAY=<server>:<graphical card>.<screen>
```

In many case it is just

```
export DISPLAY=:0 (or :1)
```

after that, one can launch a graphical application, for instance xterm (or gnuplot> plot sin(x)).

WINDOWS: cygwin installs a unix-compliant filesystem, and maps the C: drive on /cygdrive/c. One can "go" to the standard documents directory with

```
cd /cygdrive/c/Users/<franco>/Documents
```

or make a link

```
ln -s /cygdrive/c/Users/<franco>/Documents/mydir .
```

The compilation phase

source (text):
hello.c

```
#include <stdio.h>

int
main(void)
{
    puts("Hello, World!");
    return 0;
}
```

Preprocessor
cc -E hello.c

source with expanded
#directives

```
[lines omitted for brevity]

extern int __vsprintf_chk (char * restrict, size_t,
    int, size_t, const char * restrict, va_list);
# 493 "/usr/include/stdio.h" 2 3 4
# 2 "hello_world.c" 2

int
main(void) {
    puts("Hello, World!");
    return 0;
}
```

object code hello.o
without external functions

```
00000000 cf fa ed fe 07 00 00 01 03 00 00 00 01 00 00 00
00000010 04 00 00 00 08 02 00 00 20 00 00 00 00 00 00
00000020 19 00 00 00 35 01 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 98 00 00 00 00 00 00 28 02 00 00 00 00 00 00
00000050 98 00 00 00 00 00 00 07 00 00 00 00 07 00 00
00000060 04 00 00 00 00 5f 5f 74 65 78 74 00 00 00 00
00000070 00 00 00 00 00 00 00 5f 5f 54 45 58 54 00 00
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 28 00 00 00 00 00 28 02 00 00 04 00 00 00 00
000000a0 c9 02 00 02 00 00 00 04 00 00 00 00 00 00
000000b0 00 00 00 00 00 00 5f 5f 63 73 74 72 69 6e
000000c0 67 00 00 00 00 00 5f 5f 54 45 58 54 00 00
000000d0 00 00 00 00 00 00 28 00 00 00 00 00 00 00 00
000000e0 0e 00 00 00 00 5b 02 00 00 00 00 00 00 00
000000f0 00 00 00 00 00 02 00 00 00 00 00 00 00 00
00000100 00 00 00 00 00 5f 5f 63 6f 6d 70 61 63
00000110 74 5f 75 6e 77 69 6e 64 5f 5f 4c 44 00 00 00
00000120 00 00 00 00 00 00 00 3b 00 00 00 00 00 00
00000130 20 00 00 00 00 00 60 02 00 00 03 00 00 00
00000140 d0 02 00 00 01 00 00 00 00 02 00 00 00 00
00000150 00 00 00 00 00 5f 5f 65 68 5f 66 72 61
00000160 6d 65 00 00 00 5f 5f 54 45 58 54 00 00
00000170 00 00 00 00 00 58 00 00 00 00 00 00 00 00
```

Assembly
cc -c hello.c

assembly code:
hello.s

```
.section __TEXT,__text,regular,pure_instructions
.macosx_version_min 10, 10
.globl _main
.align 4, 0x90
_main:
    .cfi_startproc
## BB#0:
    pushq %rbp
    Ltmp0:
        .cfi_def_cfa_offset 16
    Ltmp1:
        .cfi_offset %rbp, -16
        movq %rsp, %rbp
    Ltmp2:
        .cfi_def_cfa_register %rbp
        subq $16, %rsp
        leaq L_str(%rip), %rdi
        movl $0, -4(%rbp)
```

Compiler
cc -S hello.c

linker
cc -c hello.c
-o hello
-l<libraries>

executable code:
hello (or a.out)

exec

Useful unix commands

ls: list files

```
macinino:Lesson1 franco$ ls
Lesson1.pptx  hello.c      prova.c
hello         hello.o      ~$Lesson1.pptx
```

ls -l (long listing)

```
macinino:Lesson1 franco$ ls -l
total 664
-rw-r--r--@ 1 franco  staff  306857 23 Feb 07:01 Lesson1.pptx
-rwxr-xr-x  1 franco  staff   12556 22 Feb 10:27 hello
-rw-r--r--@ 1 franco  staff    113 22 Feb 10:13 hello.c
-rw-r--r--  1 franco  staff    776 22 Feb 10:16 hello.o
-rw-r--r--@ 1 franco  staff    57 22 Feb 10:16 prova.c
-rw-r--r--@ 1 franco  staff    165 23 Feb 09:49 ~$Lesson1.pptx
```

```
macinino:Lesson1 franco$ ls -lt
total 664
-rw-r--r--@ 1 franco  staff    165 23 Feb 09:49 ~$Lesson1.pptx
-rw-r--r--@ 1 franco  staff  306857 23 Feb 07:01 Lesson1.pptx
-rwxr-xr-x  1 franco  staff   12556 22 Feb 10:27 hello
-rw-r--r--  1 franco  staff    776 22 Feb 10:16 hello.o
-rw-r--r--@ 1 franco  staff    57 22 Feb 10:16 prova.c
-rw-r--r--@ 1 franco  staff    113 22 Feb 10:13 hello.c
```

UNIX flags

```
macinino:Lesson1 franco$ ls -l
total 1392
-rw-r--r--@ 1 franco  staff  610557 23 Feb 15:23 Lesson1.pptx
-rwxr-xr-x  1 franco  staff    125 23 Feb 16:14 compile.sh  <- this is executable
-rwxr-xr-x  1 franco  staff  12556 23 Feb 16:14 hello      <- this is executable
-rw-r--r--@ 1 franco  staff    113 22 Feb 10:13 hello.c
-rw-r--r--  1 franco  staff    776 22 Feb 10:16 hello.o
-rw-r--r--  1 franco  staff  27806 23 Feb 10:02 hello.s
-rwxr-xr-x  1 franco  staff  12556 23 Feb 16:14 numbers  <- this is executable
-rw-r--r--  1 franco  staff    113 23 Feb 15:25 numbers.c
-rw-r--r--  1 franco  staff     4 23 Feb 15:37 pippo.dat
-rwxr-xr-x  1 franco  staff  4248 23 Feb 16:14 prova    <- this is executable
-rw-r--r--@ 1 franco  staff    57 22 Feb 10:16 prova.c
-rw-r--r--@ 1 franco  staff    165 23 Feb 09:49 ~$Lesson1.pptx
```

u/g/o: user/group/others

r = read

w = write

x = execute (or change dir, for dirs)

to change flags:

chmod o-r <file> : others cannot read

chmod +x <file> : file becomes executable (useful only if file is a script or a dir)

chmod o-x <dir> : others cannot list dir (default)

Useful unix commands

pwd (print working directory)

```
macinino:Lesson1 franco$ pwd
/Users/franco/Documents/Didattica/LFC/2019-2020/Lesson1
```

type content of a file

less hello.s

```
# 1 "hello.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 362 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "hello.c" 2

# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
per/SDKs/MacOSX.sdk/usr/include/stdio.h" 1 3 4
# 64 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Devel
oper/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
per/SDKs/MacOSX.sdk/usr/include/_stdio.h" 1 3 4
# 68 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Devel
oper/SDKs/MacOSX.sdk/usr/include/_stdio.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
per/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 1 3 4
# 630 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Deve
loper/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
hello.s
```

commands of less:

h: help

arrows (or j,k) scroll

/pattern search for the pattern

1G go to beginning (line 1)

xG go to line x

0G go to end

q exit

Useful unix commands

pwd (print working directory)

```
macinino:Lesson1 franco$ pwd
/Users/franco/Documents/Didattica/LFC/2019-2020/Lesson1
```

type content of a file: `cat hello.s`

with control: `less hello.s`

```
# 1 "hello.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 362 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "hello.c" 2

# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
per/SDKs/MacOSX.sdk/usr/include/stdio.h" 1 3 4
# 64 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Devel
oper/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
per/SDKs/MacOSX.sdk/usr/include/_stdio.h" 1 3 4
# 68 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Devel
oper/SDKs/MacOSX.sdk/usr/include/_stdio.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
per/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 1 3 4
# 630 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Deve
loper/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 3 4
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Develo
hello.s
```

commands of more:

h: help

arrows (or j,k) scroll

/pattern search for the pattern

1G go to beginning (line 1)

xG go to line x

0G go to end

q exit

Useful unix commands

redirect (concatenate): `cat hello.s > hello-copy.s`

(copies file, destroying hello-copy.s if present, one can use also `cp hello.s hello-copy.s`)

appending: `cat hello.s >> hello-copy.s`

pipe: `cat hello.s | cat > hello-copy.s`

(useless in this case)

rename (move): `mv <file> <dest>`

pipe + screen: `cat hello.s | tee hello-1.s`

removing files: `rm hello-1.s`

destroying: `rm -r <dir>`

changing dir: `cd <dir>` (back 1 level: `cd ..`)

echo: `echo 3 4 => 3 4` (like cat, but does not interpret arguments as filenames)

Unix commands

When you type a command in a terminal the shell (generally bash) does:

1. expands wildcards (`ls *.c -> ls hello.c numbers.c`) unless they are in single quote or escaped:
`ls '*c' or ls *.c => ls: *.c: No such file or directory`
2. expands shell variables like `$PATH`:
`echo $PATH =>`
`/opt/local/bin:/opt/local/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin:/opt/X11/bin`
3. parses the arguments (divides them on whitespaces, if not escaped and if not between quotes (see below))

Unix commands

4. if it is an internal command (like "ls") executes it (there are shell like busybox where most of commands are internal)
5. otherwise search <command> in the directories listed in the environmental variable \$PATH (unless the full path is specified)
6. it also looks if the file is text it is executable and begins with #!<shellpath> is passes it to shellpath (so if the file starts with #!/bin/bash and has list of commands for bash, it is executed by the shell – a script). Otherwise executes the command
7. the current directory (.) generally is NOT in the path (for security reason)

Bash scripts

Bash has a rich script language, for instance

```
macinino:~$ for i in 1 2 3 4 5; do echo $i; done  
1  
2  
3  
4  
5
```

where `i` is a variable. You can set variables using `=`

`myvar=hello.c` (or `myvar="hello.c"`)

`echo $myvar => hello.c` (also `echo ${myvar}`)

and you can take/cut/replace pieces of vars

`${myvar:3} => lo.c` `${myvar:3:2} => lo`

`${myvar%.c} => hello` (also `${myvar%.*}`)

`${myvar#hello.} => c` (also `${myvar#*.}`)

ecc. ecc.

Useful unix commands

For instance: compile all .c files in a dir

```
macinino:Lesson1 franco$ for source in *.c; do target=${source%.*}; \  
> echo "cc $source -o $target"; done  
cc hello.c -o hello  
cc numbers.c -o numbers  
cc prova.c -o prova  
macinino:Lesson1 franco$
```

the same as a script (remember to chmod +x compile.sh)

```
macinino:Lesson1 franco$ cat compile.sh  
#!/bin/bash  
  
for source in *.c  
do  
    target=${source%.c}  
    command="cc $source -o $target"  
    echo $command  
    $command  
done
```

streamhandle

In unix files and stream are almost the same

programs take input from STDIN (/dev/stdin), outputs to STDOUT (/dev/stdout) and send error to STDERR (/dev/stderr)

One can redirect STDIN with "<" es.

cat < hello.c (the same as cat hello.c)

STDOUT with ">"

STDERR with "2>"

redirecting STDOUT to STDERR

./program >&2 (it means send STDOUT (>) to address of descriptor 2 (stderr))

redirecting STDERR to STDOUT

./program 2>&1

for redirecting both to the same file

./program > file 2>&1 (first assign stdout to file, then copy stderr)

Processes

kill a process in foreground : `<control-c>`

stop a process `<control-z>`

list all processes: `ps`

list user process: `jobs`

kill a process: `kill <PID>` or `kill %1` (first stopped one)

start a process in background: `./program &`

bring a background process in foreground: `fg (fg %2)`

put a stopped process in background: `bg`

If you want to leave the process in background also when disconnecting the terminal, use `nohup`

Commandline arguments

Write a program that sums numbers on commandline

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char ** argv)
{
    double tot = 0;

    while (argc-- > 1) {
        char * endptr;
        tot += strtod(argv[argc], &endptr);
        if (*endptr) {
            fprintf(stderr,
                "trailing characters after argument #%d ('%s')\n",
                argc, endptr);
        }
    }
    printf("%lf\n", tot);
    return (0);
}
```

Commandline arguments

Write a program that sums numbers on commandline

```
macinino:Lesson1 franco$ ./sumall 3 4pluto 5e3  
trailing characters after argument #2 ('pluto')  
5007.000000
```

suppressing errors

```
macinino:Lesson1 franco$ ./sumall 3 4pluto 5e3 2>/dev/null  
5007.000000
```

gnuplotting

let's write a program that lists the first 100 numbers and its square (call it numbers.c)

```
#include <stdio.h>
int main() {
    for (int i=0; i<100; i++) {
        printf("%d %d\n", i, i*i);
    }
    return (0);
}
```

notice that main() is a function returning an int (zero usually).

compile with `cc numbers.c -o numbers`

redirect the output to a file with

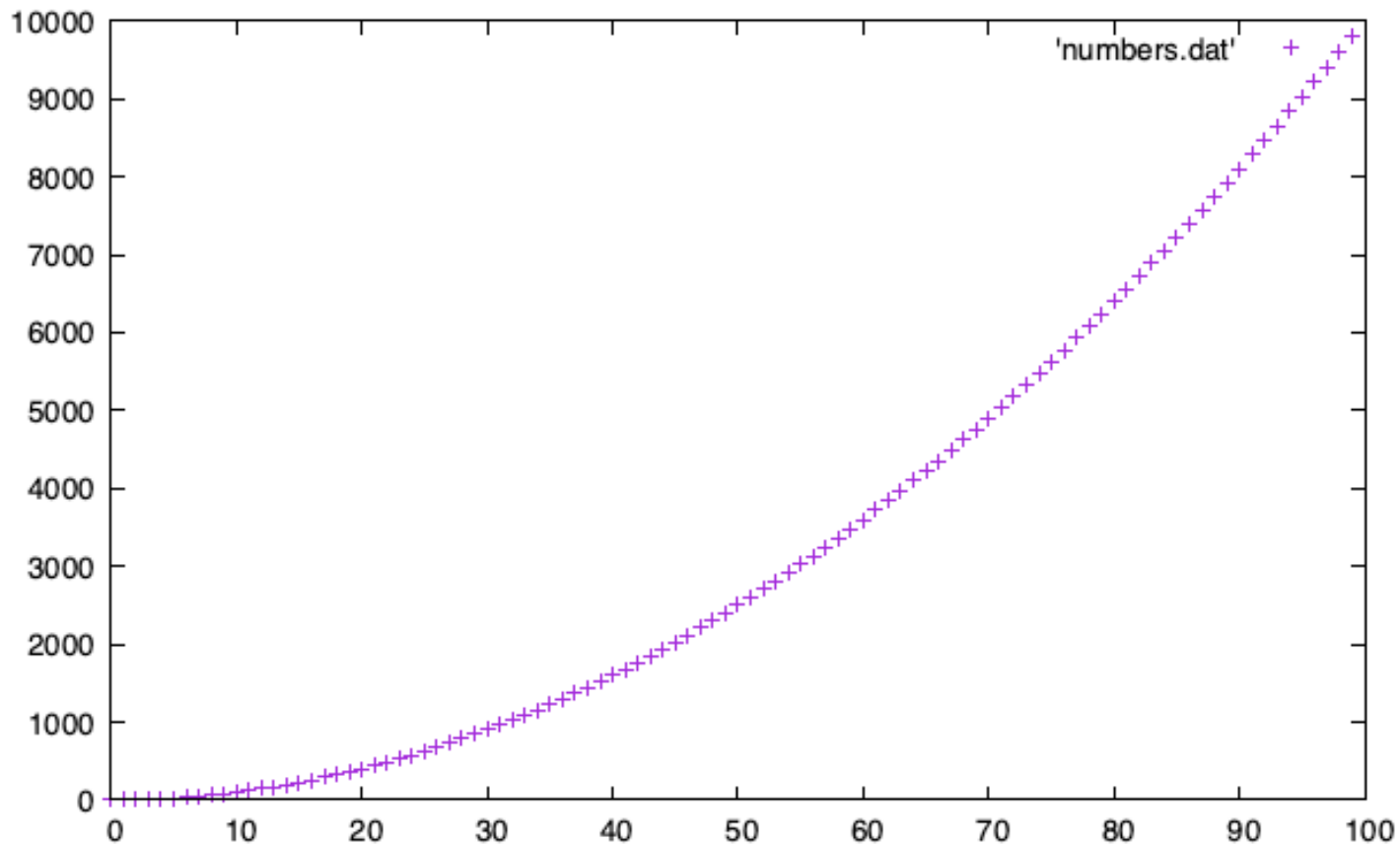
```
./numbers > numbers.dat
```

the `./` is necessary

gnuplotting

Now plot the data with gnuplot

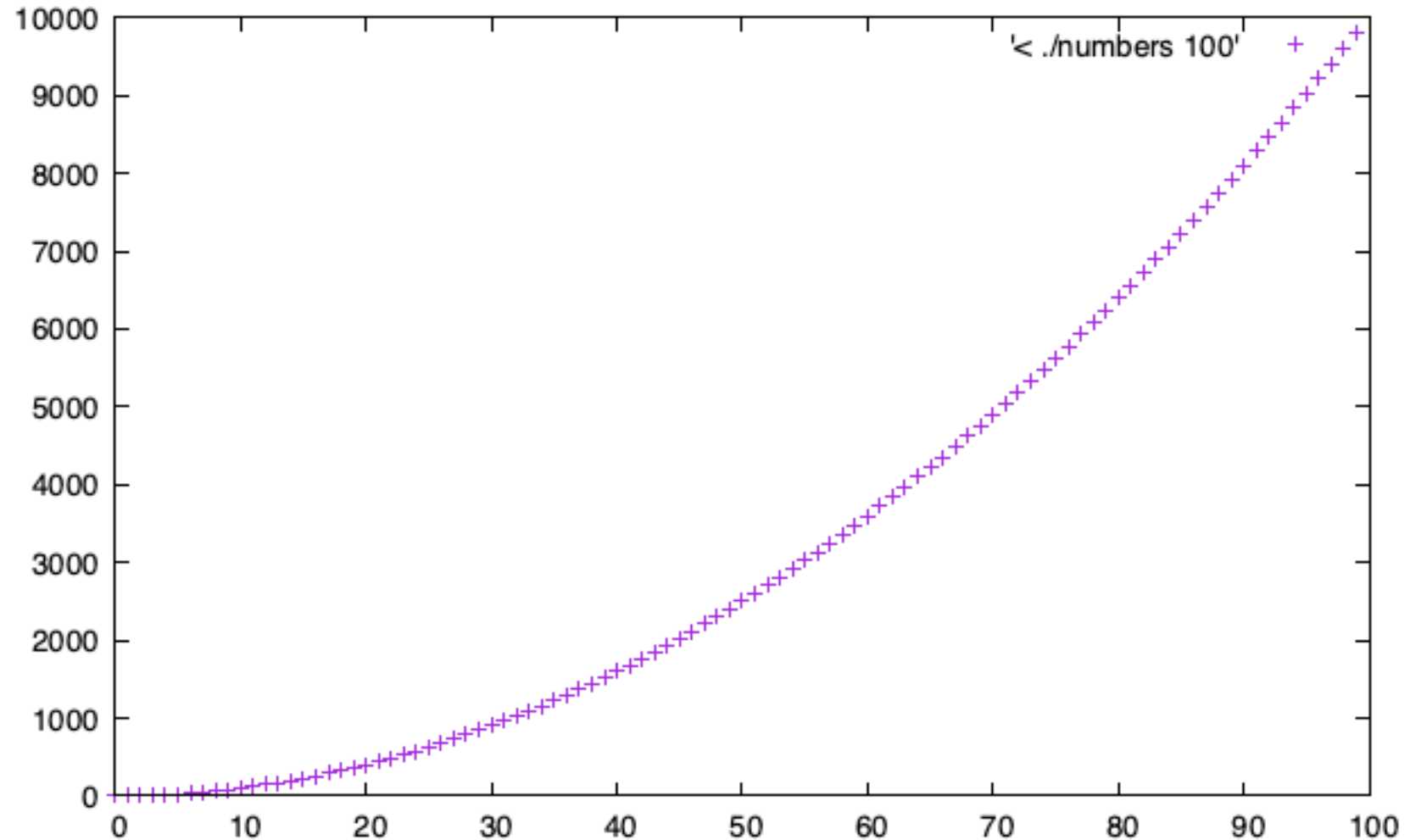
gnuplot 'numbers.dat'



gnuplotting

but one can avoid the file numbers.dat

```
gnuplot> plot "<./numbers 100"
```

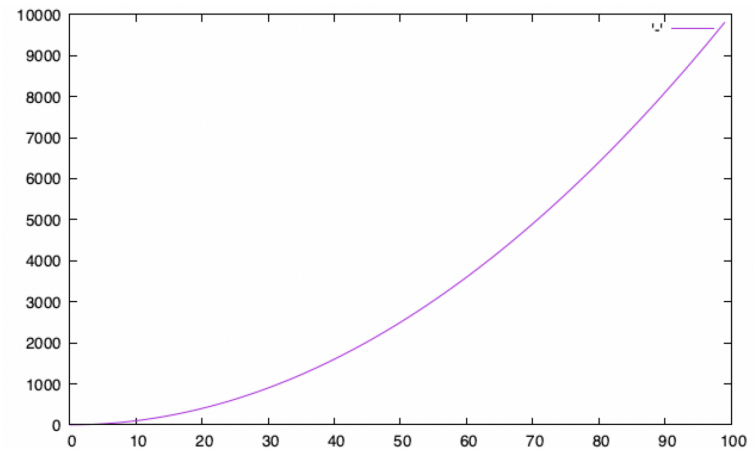


gnuplotting

one can also open (and send commands) to gnuplot from the program (useful for continuously changing

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

int main(int argc, char ** argv) {
    FILE *gp;
    if (argc < 2) {
        fprintf(stderr, "usage: %s <N>\n", argv[0]);
        return(1);
    }
    gp = popen("gnuplot", "w");
    fprintf(gp, "plot '-' w l\n");
    double N = atof(argv[1]);
    for (double x=0; x<N; x++) {
        fprintf(gp, "%.0f %.0f\n", x, x*x);
    }
    fprintf(gp, "end\n");
    fflush(gp); // empty buffer
    getchar(); // wait
    return (0);
}
```



gnuplotting

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

#define N 3

int main(int argc, char ** argv) {
    FILE *gp;
    int i;
    double x[N], y[N], vx[N], vy[N];
    double d, dt=0.1;
    gp = popen("gnuplot", "w");
    fprintf(gp, "set xrange [-10:10]; set yrange [-10:10]\n");
    fprintf(gp, "unset key; unset colorbox; set size square\n");
    for (i=0; i<N; i++) {
        x[i] = 3+i; y[i] = 0;
        vx[i] = 0; vy[i] = 0.07*x[i];
    }
    for (double t=0; ; t+=dt) {
        fprintf(gp, "plot '-' u 1:2:3 w p palette ps 3 pt 7\n");
        fprintf(gp, "0 0 3 \n"); // sun
        for (i=0; i<N; i++) {
            fprintf(gp, "%f %f %d\n", x[i], y[i], 2-i);
            x[i] += vx[i]*dt;
            y[i] += vy[i]*dt;
            d = pow(x[i]*x[i]+y[i]*y[i], 3./2);
            vx[i] -= 0.05*x[i]/d;
            vy[i] -= 0.05*y[i]/d;
        }
        fprintf(gp, "end\n");
    }
    fflush(gp); // empty buffer
    getchar(); // wait
    return (0);
}
```

