

Pencil Code: Quick Start guide

Illa R. Losada, Michiel Lambrechts, Elizabeth Cole, Philippe Bourdin

June 4, 2017

Contents

1	Required software	2
1.1	LINUX	2
1.2	MACOS X	2
2	Download the Pencil Code	2
3	Configure the shell environment	3
4	Your frist simulation run	3
4.1	Create a new run-directory	3
4.2	Linking to the sources	3
4.3	Makefile and parameters	3
4.3.1	Single-processor	4
4.3.2	Multi-processor	4
4.4	Compiling...	4
4.4.1	Using a different compiler (optional)	4
4.4.2	Changing compiler options (optional)	5
4.5	Running...	5
4.6	Troubleshooting...	5
5	Data post-processing	6
5.1	IDL visualization (optional, recommended)	6
5.1.1	GUI-based visualization (recommended for quick inspection)	6
5.1.2	Command-line based processing of “big data”	6
5.1.3	Command-line based data analysis (may be inefficient)	6
5.2	Python visualization (optional)	7
5.2.1	Python module requirements	7
5.2.2	Using the 'pencil' module	7

1 Required software

1.1 LINUX

A FORTRAN and a C compiler are needed to compile the code. Both compilers should belong to the same distribution package and version (e.g. GNU GCC 4.8.4, 64 bit Linux).

1.2 MACOS X

For MAC, you first need to install XCODE from the website <http://developer.apple.com/>, where you have to register as a member. Alternatively, an easy to install gfortran binary package can be found at the website <http://gcc.gnu.org/wiki/GFortranBinaries>. Just download the archive and use the installer contained therein. It installs into `/usr/local/gfortran` with a symbolic link in `/usr/local/bin/gfortran`. It might be necessary to add the following line to the `“.cshrc”`-file in your `/home` folder:

```
setenv PATH /usr/local/bin:\$PATH
```

2 Download the Pencil Code

The Pencil Code is an open source code written mainly in FORTRAN and available under GPL. General information can be found at our official homepage:

<http://pencil-code.nordita.org/>.

The latest version of the code can be downloaded with `svn`. In the directory where you want to put the code, type:

```
svn checkout https://github.com/pencil-code/pencil-code/trunk/ pencil-code
```

Alternatively, you may also use `git`:

```
git clone https://github.com/pencil-code/pencil-code.git
```

More details on download options can be found here: <http://pencil-code.nordita.org/download.php>

The downloaded `'pencil-code'` directory contains several sub-directories:

1. `'doc'`: you may build the latest manual as PDF by issuing the command `make` inside this directory
2. `'samples'`: contains many sample problems
3. `'config'`: has all the configuration files
4. `'src'`: the actual source code
5. `'bin'` and `'lib'`: supplemental scripts
6. `'idl'`, `'python'`, `'julia'`, etc.: data processing for diverse languages

3 Configure the shell environment

For a quick start, you need to load some environment variables into your shell. First, you enter to the freshly downloaded directory:

```
cd pencil-code
```

Depending on which shell you use, you can do that by a simple command:

```
. sourceme.sh
```

that will work for bash and all sh-compatible shells, while this command:

```
source sourceme.csh
```

is for tcsh and any csh-compatible shell.

4 Your frist simulation run

4.1 Create a new run-directory

For a quick start, you simply create a run-directory and clone the input and configuration files from one of the samples that fits you best to get started quickly (here ‘1d-tests/jeans-x’):

```
mkdir -p /data/myuser/myrun/src
cd /data/myuser/myrun
cp $PENCIL_HOME/samples/1d-tests/jeans-x/*.in ./
cp $PENCIL_HOME/samples/1d-tests/jeans-x/src/*.local src/
```

Your run should be put outside of your ‘/home’ directory, if you expect to generate a lot of data and you have a tight storage quota in your ‘/home’.

4.2 Linking to the sources

One command sets up all needed symbolic links to the original Pencil Code directory:

```
pc_setupsrc
```

4.3 Makefile and parameters

Two basic configuration files define a simulation setup: “src/Makefile.local” contains a list of modules that are being used, and “src/cparam.local” defines the grid size and the number of processors to be used. Take a quick look at these files...

4.3.1 Single-processor

An example using the module for only one processor would look like:

```
MPICOMM=nompicomm
```

For most modules there is also a “no”-variant which switches that functionality off.

In “src/cparam.local” the number of processors needs to be set to 1 accordingly:

```
integer, parameter :: ncpus=1,nprocx=1,nprocy=1,nprocz=ncpus/(nprocx*nprocy)
integer, parameter :: nxgrid=128,nygrid=1,nzgrid=128
```

4.3.2 Multi-processor

If you like to use MPI for multi-processors simulations, be sure that you have a MPI library installed and change “src/Makefile.local” to use MPI:

```
MPICOMM=mpicomm
```

Change the `ncpus` setting in “src/cparam.local”. Think about how you want to distribute the volume on the processors — usually, you should have 128 grid points in the x-direction to take advantage of the SIMD processor unit. For compilation, you have to use a configuration file that includes the “_MPI” suffix, see below.

4.4 Compiling...

In order to compile the code, you can use a pre-defined configuration file corresponding to your compiler package. E.g. the default compilers are `gfortran` together with `gcc` and the code is being built with default options (not using MPI) by issuing the command:

```
pc_build
```

Alternatively, for multi-processor runs (still using the default GNU-GCC compilers):

```
pc_build -f GNU-GCC_MPI
```

4.4.1 Using a different compiler (optional)

If you prefer to use a different compiler package (e.g. using `ifort` or with MPI support), you may try:

```
pc_build -f Intel
pc_build -f Intel_MPI
pc_build -f Cray
pc_build -f Cray_MPI
```

More pre-defined configurations are found in the directory “pencil-code/config/compilers/*.conf”.

4.4.2 Changing compiler options (optional)

Of course you can also create a configuration file in any subdirectory of ‘pencil-code/config/hosts/’. By default, `pc_build` looks for a config file that is based on your `host-ID`, which you may see with the command:

```
pc_build -i
```

You may add your modified configuration with the filename “`host-ID.conf`”, where you can change compiler options according to the Pencil Code manual. A good host configuration example, that you may clone and adapt according to your needs, is “`pencil-code/config/hosts/IWF/host-andromeda-GNU_Linux-Linux.conf`”.

4.5 Running...

The initial conditions are set in “`start.in`” and the parameters for the main simulation run can be found in “`run.in`”. In “`print.in`” you can choose which quantities are written to the file “`data/time_series.dat`”.

Be sure you have created an empty ‘`data`’ directory. It is now time to run the code:

```
mkdir data
pc_run
```

Welcome to the world of Pencil Code!

4.6 Troubleshooting...

If compiling fails, please try the following — with or without the optional `_MPI` for MPI runs:

```
pc_build --cleanall
pc_build -f GNU-GCC_MPI
```

If some step still fails, you may report to our mailing list: <http://pencil-code.nordita.org/contact.php>. In your report, please state the exact point in this quick start guide that fails for you (including the full error message) — and be sure you precisely followed all non-optional instructions from the beginning.

In addition to that, please report your operating system (if not LINUX-based) and the shell you use (if not `bash`). Also please give the full output of these commands:

```
bash
cd path/to/your/pencil-code/
source sourceme.sh
echo $PENCIL_HOME
ls -la $PENCIL_HOME/bin
cd samples/ld-tests/jeans-x/
gcc --version
gfortran --version
pc_build --cleanall
pc_build -d
```

If you plan to use MPI, please also provide the full output of:

```
mpicc --version
mpif90 --version
mpiexec --version
```

5 Data post-processing

5.1 IDL visualization (optional, recommended)

5.1.1 GUI-based visualization (recommended for quick inspection)

The most simple approach to visualize a cartesian grid setup is to run the Pencil Code GUI and to select the files and physical quantities you want to see:

```
IDL> .r pc_gui
```

If you miss some physical quantities, you might want to extend the two IDL routines `pc_get_quantity` and `pc_check_quantities`. Anything implemented there will be available in the GUI, too.

5.1.2 Command-line based processing of “big data”

Please check the documentation inside these files:

“pencil-code/idl/read/pc_read_var_raw.pro”	efficient reading of raw data
“pencil-code/idl/read/pc_read_subvol_raw.pro”	reading of sub-volumes
“pencil-code/idl/read/pc_read_slice_raw.pro”	reading of any 2D slice from 3D snapshots
“pencil-code/idl/pc_get_quantity.pro”	compute physical quantities out of raw data
“pencil-code/idl/pc_check_quantities.pro”	dependency ckecking of physical quantities

in order to read data efficiently and compute quantities in physical units.

5.1.3 Command-line based data analysis (may be inefficient)

Several IDL-procedures have been written (see in ‘pencil-code/idl’) to facilitate inspecting the data that can be found in raw format in ‘jeans-x/data’. For example, let us inspect the time series data

```
IDL> pc_read_ts, obj=ts
```

The structure `ts` contains several variables that can be inspected by

```
IDL> help, ts, /structure
** Structure <911fa8>, 4 tags, length=320, data length=320, refs=1:
IT                LONG          Array[20]
T                 FLOAT         Array[20]
UMAX              FLOAT         Array[20]
RHOMAX            FLOAT         Array[20]
```

The diagnostic UMAX, the maximal velocity, is available since it was set in “jeans-x/print.in”. Please check the manual for more information about the input files.

We plot now the evolution of UMAX after the initial perturbation that is defined in “start.in”:

```
IDL> plot, ts.t, alog(ts.umax)
```

The complete state of the simulation is saved as snapshot files in “jeans-x/data/proc0/VAR*” every dsnap time units, as defined in “jeans-x/run.in”. These snapshots, for example “VAR5”, can be loaded with:

```
IDL> pc_read_var, obj=ff, varfile="VAR5", /trimall
```

Similarly tag_names will provide us with the available variables:

```
IDL> print, tag_names(ff)
T X Y Z DX DY DZ UU LNRHO POTSELF
```

The logarithm of the density can be inspected by using a GUI:

```
IDL> cslice, ff.lnrho
```

Of course, for scripting one might use any quantity from the ff structure, like calculating the average density:

```
IDL> print, mean(exp(ff.lnrho))
```

5.2 Python visualization (optional)

Be advised that the PYTHON support is still not complete or as feature-rich as for IDL.

5.2.1 Python module requirements

For this example we use the modules: numpy and matplotlib.

5.2.2 Using the 'pencil' module

After sourcing the “sourceme.sh” script (see above), you should be able to import the pencil module:

```
import pencil as pc
```

Some useful functions:

“pc.read_ts”	read “time_series.dat” file. Parameters are added as members of the class
“pc.read_slices”	read 2D slice files and return two arrays: (nslices, vsize, hsize) and (time)
“pc.animate_interactive”	assemble a 2D animation from a 3D array