Pencil Code: Quick Start guide

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

February 17, 2015

Contents

1 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 http://pencil-code.googlecode.com/svn/trunk/ pencil-code
```

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

2 Configure the shell

For a quick start, you need to load some environment variable 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.

3 Fortran

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 4.8.3, 64 bit Linux).

3.1 Fortran on a mac machine

For Mac, you first need to install `Xcode` from the `AppleDeveloper` site `http://developer.apple.com/`. This requires you to first register as a member. An easy to install `gfortran` can be found at `http://gcc.gnu.org/wiki/GFortranBinaries`. Just download it and it comes with an installer. It installs in the directory `/usr/local/gfortran` with a symbolic link in `/usr/local/bin/gfortran`. It might be necessary to add the following line in the `.cshrc`-file in the home folder:

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

4 Try a sample

Go to a folder that contains one of the many available samples, e.g.:

```
cd samples/1d-tests/jeans-x
```

You may also start with a fresh directory and copy over the files from one of the samples.

4.1 Setting up...

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

```
pc_setupsrc
```

4.2 Makefile

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...

4.2.1 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
```

4.2.2 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.

4.3 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 by issuing the command:

```
pc_build
```

4.3.1 Using a different compiler (optional)

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

```
pc_build -f compilers/Intel
pc_build -f compilers/GNU-GCC_MPI
```

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

4.3.2 Chaning 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.

4.4 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 physical quantities are written to the file `data/time_series.dat`.

It is time to run the code with — be sure you have an empty `data` directory:

```
mkdir data
pc_run
```

Welcome to the world of Pencil Code! Visualizing the output can be done with `IDL` or `Python`, see below.

4.5 IDL visualization (optional)

4.5.1 GUI-based visualization

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.

4.5.2 Command-line based and scripting

Several `idl`-procedures have been written (you can find them in `pencil-code/idl`) to facilitate inspecting the data (which can be found in raw format in `jeans-x/data` directory). 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> print, tag_names(ts)
IT T UMAX RHOMAX
```

The diagnostic `UMAX`, the maximal velocity, is available since it was set in `jeans-x/print.in` (more on diagnostic output can be found in section 5.4 in the manual). We can now plot the evolution of the maximal velocity in time

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

after the initial perturbation we inserted in `start.in`.

The complete state of the simulation, a snapshot, is saved (as `jeans-x/data/proc0/VAR$\dots@`, every `dsnap` time units (see `jeans-x/run.in`). These states can be inspected, for example

```
IDL> pc_read_var, obj=ff, ivar=1, /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
```

so the logarithm of the density in the simulated domain can be inspected by

```
IDL> plot, ff.lnrho
```

4.6 Python visualization (optional)

4.6.1 Python module requirements

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

4.6.2 Using the 'pencil' module

After executing 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.