Guidelines, tips and assessment criteria for the tasks of ScS

Emil Loevbak, Joris Tavernier & Pieter Appeltans *October 1, 2018*

Introduction

These criteria are based on the feedback sheet [1], the tips in [2], last year's guidelines and general remarks with what's been sent in last years. These can of course be overruled by other communications. With the assessment criteria further in this document, what's optional is indicated with square brackets.

Setup and activating Fortran compilers

Log in with Ubuntu, search (icon left above) for 'terminal', drag it to the task bar and click.

Open `~/.options.sh` (via `$ gedit ~/.options.sh` in the terminal or in map representation Edit-Preferences-Views-Show hidden and backup files) and add `CS_FORTRAN=yes`. Possibly restart the terminal.

or

Press `Ctrl-Alt-F1` and log in onto the text console. Execute the command `echo CS_FORTRAN=yes >> ~/.options.sh` and log out (`logout`). Press `Ctrl-Alt-F7` or `Ctrl-Alt-F8` and log in onto the graphical screen.

ssh

You can use your own laptop; however, installing the necessary compilers (ifort is free for students via `https://software.intel.com/en-us/qualify-for-free-software/student`) on it will be difficult. So rather work with ssh. You can use your own laptop in the Software design Labs or work from home in Linux with for example `ssh s0000000@ssh.cs.kuleuven.be` or with `st.cs.kuleuven.be` and then `ssh -X -l s0000000 Y.cs.kotnet.kuleuven.be` with Y a name of a machine in the computer classes. In Windows, you can install for example Putty. As pasword, you take either the one belonging to your student number `s0000000`, or one you create: see the documentation of the system management, for example `http://system.cs.kuleuven.be/cs/system/wegwijs/computerklas/internet/index-E.shtml`.

You have to enable X11 forwarding for graphic applications (ex gedit) and always start XMing. However, this is very slow; you can use `vim file.f95` to work in the terminal. If you think you 'll work with ssh, try to get this working in time (not one day before a deadline) using the documentation of system management, someone who already got it working, ...

And, thanks to Michiel and Willem: *To establish a ssh connection with the PC-room, it is useful to follow this tutorial: `http://dietercastel.com/2013/03/02/ssh-to-computer-science/`. Among others, this clearly mentions how to generate a keypair, then having to connect with a buffer server to connect to a computer of the PC room from there. However, one should notice that the SSH keys only work after one day.*

I did not test the scripts for transferring files from and to your own map on the server. Instead of these scripts, I use the program MobaXterm replacing Putty to make the ssh connection. You can start a new session in this by clicking on session→ssh and then doing the same as the blog indicates for Putty (first connect with the buffer server st.cs.kuleuven.be, then with a PC room computer via the command: ssh $rxxxxxxx@orval.cs.kotnet.kuleuven.be or replace orval with another Pc room computer). You can also just load a saved Putty profile to connect with the buffer server so that MobaXterm immediately finds your private key via Putty.

To send files, you also start a new session, but choose SFTP. Then fill in your r number as user name and as host you take for example: orval.cs.kotnet.cs.kuleuven.be. Another name of a PC room computer obviously also works. You can then just drag files between your own computer, left in the window, and the server to the right.

Useful terminal commands

`$ cd ~/`, `$ cd ..`, `$ ls`, `pwd`, `$ man {command}` , `$ {commando} --version`, `$ pdflatex source.tex`, `$ make`, `$ make r FC=g95`, `$ mv file dir/`, `$ cp file dir/`, `$ mkdir dir`, `$ top`, `$ locate hello`, ↑, Tab (automatic completion), Alt-Tab, Ctrl-s, Ctrl-c (interruption of program execution), Ctrl(-Shift)-c/v, Alt-↑, `$ time {prog}`, `$ valgrind {prog}`, `$ cat {in} | {prog}`, `$ ./prog < {in} > {out}` , `$ aspell check -t --lang=en report.tex`, `$ ulimit -v 1000`, `$ lscpu`, `$ free`,

Handing in tasks

For the exercise sessions, you should hand in the electronic version(s) of your source file(s) that are asked in the assignment for the task: see further. The required figure(s) should be included. Make sure they have other file names than previously handed in tasks and do not send `*.zip` or equivalent. So, you do not have to make a report in pdf (unless explicitly stated in the assignment for that task). The way of handing in the tasks will be mentioned in the assignment.

The deadline for submitting the electronic version of a task will be communicated when it is distributed. In general, there will be a deadline every one or two weeks. Deadlines are strict, both now as in your professional career; see for example 'Tasks-Varia-Deadline'.

Add the following for each file in comments at the top:

- Your name.
- Which compiler(s) (+ version number) you tested your program with *on a machine in the computer class.*
- Which compiler command (preferrably one) and which compiler options should be used to compile your program. A structured `Makefile` is also an option as an extra attachment when there are many commands (> 3 per compiler).
- Answers and output for the questions, in comments at the top of the respective source file.
- See 'Tasks'.

In addition for the first file in attachment:

- How much time you spent on the task itself in total, possibly also on the preparation of the exercise session separately but not on (catching up) the exercise session itself. So try to keep an estimate while making the task. The time spendings that are sent in only serve as a global weighing between the heaviness of the tasks.
- Preferrably, this is the main program.

Supplement of the rules

All tasks for the course Scientific Software are graded, and the exam rules therefore apply. For the sake of completeness, we describe what is whether or not allowed in terms of cooperation below.

The solution and/or report and/or program code that are handed in, have to fully be the result of work you have performed yourself. You can of course discuss with other students, in the sense that you talk about general solution methods or algorithms, but the discussion cannot be about specific code or report text that you are writing, nor about specific results that you wish to hand in. If you talk with others about your tasks, this can NEVER lead to that you are at any time in possession of a whole or partial copy of the solved task or report of others, regardless of the fact whether that code or report is on paper or available in electronic form, and independent of who wrote the code or report (fellow students, possibly from other study years, complete outsiders, internet sources, etc.). This also encompasses that there is no valid reason at all to pass your code or report to fellow students, nor to make this available via publicly accessible directories or websites.

Every student is reponsible for the code and the work he or she hands in. If there are any doubts during the assessment of a task about the fact whether this is self made (ex. similar code, graphs, text or results), the student will be asked to give an explanation for this. If this does not eliminate the doubts, this will be reported as an irregularity, as foreseen in the education and exam regulations (see `http://www.kuleuven.be/education/regulations/index.php`).

Ronald Cools, Karl Meerbergen
P.S. Plagiarism detection software can be used to detect problems.

Tasks

Code

IO File format and structure as asked.
Read in and write out: as specified and not too much.
Print '!' in front of results so that you can just copypaste output to the top of your source file for tasks.

Precision	Choose a suited kind, preferrably parametrised, avoid compiler dependent definitions.
Code	Interface of your (sub)routines is free to choose. Document shortly and briefly if it is not trivial and improves the readability of your code. In each file at the top your name and a short description. Deallocate, `implicit none` everywhere. Completeness: implementation of all specifications.
Style	See example code on Toledo, ‘Scientific Software Development with Fortran’, the Fortran standaard itself, assignments of the exercise sessions and given examples. Not that important by itself, but be consequent. Clarity: Logic structure, no commented or duplicated code (so not too much copypasting), avoid too ‘object oriented’ (many types and small functions + difficult to read and to find the effective calculations, ...), alignment/indentation, readability, ...
Performance	[Speed (CPU, wallclock, ...), memory, accuracy (maximum norm, elementwise, 2-norm, ...), convergence speed, complexity, parallelisability, extendability, readability, ease of installation, ...] [Discussion of caching effects, loop unrolling, spatial and temporal locality, prefetching.] [Cypher material and/or graphs with their discussion.] [Optimising performance, for example by changing the order of loops/ operations, reusing memory, use of external libraries such as BLAS and/or by using pass by reference in stead of pass by value.] [Vectorised instead of elementwise with forall/indexing/intrinsic functions/...] [Static $\leftrightarrow$ dynamic memory $\leftrightarrow$ pointers, stack versus heap.] Better inline short function instead of a separate routine for speed and sometimes even for extendability. Use Lapack and BLAS, see for example `http://www.netlib.org/lapack/single/`. Think about the use and readability of the code you write.
Debugging	Discuss briefly whether there were any problems. Do not give variables the name of (intrinsic) functions/routines. Fortran (often) does not make a distinction between capital letters and small letters. Do not test floating points numbers on exact equality. When you have syntax errors, first solve the first one since the next ones could be caused by it. Initialise all variables (for example on 0) and check whether it makes a difference if you do that in the definition. A line is cut off on 80 or 132 characters. Compare with a (Matlab-)implementation or the exact result. Test every change since your backup.
Driver	Tested on all compilers: some give a better performance or a compiler error instead of runtime segmentation faults.

Algorithm

Experiments	Reliable time estimates [averaged out] + discussion. [Computational/Memory/... complexity and possibly an estimate or measurement.] Discuss impact of other factors.
Algoritm	Working out missing details, usually as a formula.
Verification	Test with a simple example. [Give enough examples and a discussion that is not too short: try special cases that try to make your program fail, other (qualitative) behavior, ...] Executing the main program should print the output you give at the top of your code, containing all tests preferably without user interaction unless specified in the assignment. [Graphs to verify the algorithm or evaluation of performance: see ‘Discussion’.]
Robustness	Test with an unknown, unexpected or special input/option/argument, empty arrays, numbers that are zero, negative or very deviant. Prevent the program from crashing, doing something that the user does not expect or waiting for input: it is better to give an error and stop. Balance this with the readability and conciseness of your code. [Also test in your code whether for example certain variables become `NaN`, bigger or smaller than expected.] Check for mathematical or physical impossibilities.

Varia

Remarks from or about the discussion at the top of the code.

Makefile (only if many commands): leave out what you do not use, do not use absolute paths (/Desktop/, /usr/, ...), give a possibility to recompile to avoid dangers while debugging. When using `call SYSTEM(...)` with nagfor, you can let your Makefile add 'use f90_unix_proc', add a short file, use `execute_command_line(...)` or find another solution.

Flags [Discussion of compile flags and their effect].

Feedback You will get individual feedback each time: try to take it into account when making the next tasks.

Deadline Everything handed in correctly and on time: no *.o, *.mod,*.out, backups (.f90∼ etc), hidden files (.gout...), executables, non-asked libraries, ...
[To be able to respect the timing, you can make backups of working versions of your program, to compare with and so that you can fall back on it when you change something that makes the program fail.]
Tasks that are handed in well in advance, generally contain less errors. This also allows correcting everyone's assignment in time. We also consider that you cannot take into account things that appear on the discussion forum after you have sent in.
Spread your time spending.
Try not to lose too much time with parts that are not that relevant so post something on the discussion forum (have a look at this regularly) if you think something is too hard or spend too much time on it.
(Often) handing in too late (without a founded reason) is discussed at the oral discussion, certainly when handing in after (general) feedback has been given.
Always re-read the assignment, the discussion forum and these guidelines before you send in to make sure you didn't forget anything.

Completeness When you think you are going to spend too much time on some part of the assignment, ask for clarification on the discussion forum. If you're stuck on a small part, ask for a solution of it via mail so that you can at least do what follows.
If you cannot carry out some specifications of the assignment, you can leave them out: do not spend hours on the same mistake! Do clearly mention it and why; it is of course necessary to implement as many specifications as possible.

Optional [Carrying out optional parts, extension of assignment/exercise session, additional optimisations of the code.]

Agreement Make sure there is agreement between comments at the top of the code and (results of) the code.

Doubt When the assignment is not clear, you ask a question on the discussion forum.

Ergonomy Try not to unnecessarily burden your body: posture, distance to the screen, screen brightness, ... `https://admin.kuleuven.be/sab/vgm/kuleuven/EN/ergonomics/Ergon`

Individual You can consult your collegues on a conceptual level, preferably on the discussion forum, but everyone makes the tasks individually.
If the task an extension to the exercise session, you also mention who you worked with for the part of the exercise session: work on different machines as much as possible.
Preferably ask questions on the discussion forum, but if you do make use of sites like stackoverflow, be reluctant in the quantity of code communicated and put the answer on the discussion forum.

Discussion

Content Avoid vague discussions without specific references to your code.
The reasoning followed is sufficiently profound and nuanced and possible suppositions are motivated enough.
If a certain part of the assignment did not work out or when your code does not compile, state this clearly in your report and indicate where you expect the problem to be.

Results and analyses are reported systematically. Only relevant results are discussed and interpreted.
Analysis and (physical) interpretation of data are complete and correct. Limitations and implications for the interpretation of the results are reported.
If you give timings, also mention whether a part of the CPU is taken by other users.

Language The language in your code documentation/what you print has to be correct written language.

Visualisation The corresponding text always and explicitly references the table, figure of graph. An interpretation is also given.
In a graph, axes are named, (physical) quantities and units are represented.
Graphs must be clear and different colors could be mixed with dashed lines etc to avoid problems when

	printing in black and white. Graphs have relevant (linear, (semi)logarithmic) and named axes. When there are many figures, think how you can put them together to compare different methods easier, for example.
Formulas	The number of digits shown is only high when necessary/relevant.
References	Give references for everything you used and that is not ‘generally known’. Certainly refer to original code if you adapted/used it (for a non-essential part of the task !). In assignments like these, it is not really required to create a separate bibliography list.
Connect	your personal approach to the knowledge obtained in the lectures, exercise sessions and when reading the course material by explicitly referring to them.

References

[1] Faculty of Engineering Science. Feedback sheet written communication. `https://eng.kuleuven.be/english/education/reporting/feedback-sheet-written-communication-2.pdf`, february 2015.

[2] D. Nuyens. Thesis layout in LaTeX. `http://people.cs.kuleuven.be/~dirk.nuyens/thesislayout/`, april 2014.