Code Standards

Daniel Celis Garza

April 19, 2016

1 Introduction

Documenting coding standards.

2 General Practices

Always use `implicit none` before any variable declaration. Include only what will be used in the code via `use <modue_name>, only : <list>`.

Use spaces to clearly denote what is going on in the code. Try not to use `**` if elevating to a low integer power. Should `**` become necessary, do not add spaces between them. Use line breaks to improve the code's readability. Align similar lines of code to make the code neater.

Modularise code and make subroutines as general as possible.

3 Files

3.1 Makefile

Makefiles should start with `make_` followed by the main file's name. They should have, *at minimum*, the following flags: array bounds, unused variables & unallocated variables. Comments should be used to explain what each file and flag is needed/used for.

3.2 Main

Main filenames must be as clear and short as possible, where clarity $\gg$ length. Use the following naming structure: `<project>_main`. The extension must be `.out` for Unix systems and `.exe` for Windows. It should only contain calls to other subroutines and functions.

3.3 Modules and Sub-modules

Do *NOT* use global variables and/or parameters unless *completely* necessary. Global parameters and variables have a tendency to bite one in the ass when one least expects it. If they seem necessary, such as in the case of user-defined parameters, use a module to read and store them so they become global variables defined at runtime.

Modules and submodules should be named under the same scheme as main files, without the underscore and what follows it, after the underscore.

Limit a module's scope to a specific purpose.

3.4 Data

Data filenames should use the either of the following naming structures: `<project>.dat` or `<project>_<parameters>.dat`.

Table 1: Data file print order.

Independent Vars	Dependent Vars
$x_{1\ldots n}$	$\dfrac{\mathrm{d}^{0\ldots k}}{\mathrm{d}x_{1\ldots n}^{0\ldots k}} f_{1\ldots m}(x_{1\ldots n})$

The top line of the data file should contain a short commented description of the file structure and purpose. Whenever possible, the column print order should be:

4 Variables and Arguments

Should a subroutine or function be used as an argument of another function or subroutine, make interfaces explicit. Use derived types, pointers and allocatable variables

Variable and subroutine names should make it clear what they mean, pointers should start with `p_`. The same principle applies as in naming files. They should be declared by type with kind in ascending order: *1*) Scalar, *2*) Array, *3*) Coarray, *4*) Optional, *5*) Derived, *6*) Pointer, *7*) Allocatable; and intent: *1*) `intent(in)`, *2*) `intent(out)`, *3*) `intent(inout)`, *4*) locals, and *5*) `parameter`.

5 Intent

`intent()` must always be declared within functions and subroutines.

6 Functions and Subroutines

Aforementioned principles apply to their naming conventions.

6.1 Elemental

Elemental functions & subroutines should start their name with `elmt_`.

6.2 Recursive

Recursive functions & subroutines should start their name with `rcsv_`.

6.3 Elemental Recursive

Elemental functions & recursive subroutines should start their names with `rcel_`.

7 Comments and Labels

Comments should be used to explain the reasoning behind logical blocks. Expanded comments should be used whenever a non-trivial procedure is being carried out.

Every loop, if and case block must be labeled with a logical label made up of the initial letters of the block's main purpose as delineated by the comment explaining its function.

Subroutines and functions purpose and variables must be documented before `implicit none`. These comments must have the a structure similar to the following:

```
!=======================!
! Short explanation.    !
! Method used.          !
! Author & date         !
!-----------------------!
! Extended explanation. !
!-----------------------!
! Inputs:               !
! scalar  = explanation !
! array() = explanation !
!-----------------------!
! Outputs:              !
!-----------------------!
! Inputs-Outputs:       !
!-----------------------!
! Locals:               !
!=======================!
```