Scientific Software
Session 2: Fortran 95, Working with numbers on a digital computer

Emil Løvbak, Wouter Baert & Pieter Appeltans October 15, 2020

"Real numbers do not exist in a computer."

Introduction

This exercise session examines some peculiarities encountered when working with numbers (both integer and real) on a digital computer. The aim of this session is for you to develop an understanding of the following concepts:

- recursive functions;
- integer overflow;
- floating point numbers in Fortran 95;
- floating point issues: catastrophic cancellation, overflow and underflow, rounding errors and division by small numbers.

Exercises

1. Last week we encountered `elemental` procedures, which are used to apply an operation element wise to an array. This week we will use `recursive` functions, which are functions that call themselves[1]. A function can be declared recursive by adding `recursive` before the specifier `function` (**Note** that for recursive functions the name of the output variable should be specified using a `result` clause and should differ from the function name). Now, make a `recursive` function that calculates the double factorial. The double factorial of an integer n is defined as follows:

$$n!! = \begin{cases} n \cdot (n-2)!! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \text{ or } n = -1 \\ 0 & \text{if } n < -1. \end{cases}$$

   For example, the double factorial of 5 is computed in the following way:

$$5!! = 5 \cdot 3!! = 5 \cdot (3 \cdot 1!!) = 5 \cdot (3 \cdot (1 \cdot (-1)!!)) = 5 \cdot (3 \cdot (1 \cdot 1)) = 15.$$

   To implement this function, use a `select case` block. Test your implementation for all integers between $n = -2$ and $n = 49$.

   - What happens?
   - Illustrate the problem using at least two different integer kinds. **Make sure that your code is compiler-independent.**

   Link preparation: Scientific Software Development with Fortran 2.12 (Conditionals); Fortran 2003 - NOTE 12.38 (An example of a recursive function); Fortran 2013 standard - 13.5.4 (Kind functions)

2. Download `print_real_properties_gfortran.f90`, `print_real_properties_g95.f90`, `print_real_properties_ifort.f90` and `print_real_properties_nagfor.f90` from Toledo and inspect the source code. These programs output the following constants for the different kind types of

[1] Note that Fortran also supports `recursive` subroutines, which work in a similar way.

real numbers: the kind type parameter, the number of bits in the mantissa, the equivalent decimal precision, the minimal and the maximal exponent, the equivalent decimal range, and the machine precision. Compare the different compilers at your disposal: compile `print_real_properties_gfortran.f90` with `gfortran`, `print_real_properties_ifort.f90` with `ifort`, etc..

- Which types of real numbers are available on the different compilers?
- Does the same kind type have the same kind type parameter on the different compilers?

Link preparation: Fortran 2003 standard - 13.5.4, 13.5.6

3. (a) Write a Fortran program that evaluates the function

$$f(x) = \frac{1 - \cos(x)}{x^2}$$

for $x = 10^{-1}, 10^{-2}, \ldots, \epsilon_{\text{mach}}$ (the machine precision) in double precision and that writes the result to the standard output stream. Note that $\lim_{x \to 0} f(x) = 1/2$. **Do you get the result you expected? Why (not)?**

(b) Using $\cos(x) = \cos^2(x/2) - \sin^2(x/2)$, it can be verified that f is equivalent to

$$g(x) = \frac{2\sin^2(x/2)}{x^2}.$$

Compare f and g for $x = 10^{-1}, 10^{-2}, \ldots, \epsilon_{\text{mach}}$. **Although f and g are analytically identical, this is not the case when evaluated numerically. Explain why?**

(c) Now repeat the experiment in quadruple precision. **How can you easily switch between different precisions (kind types)?**

Link preparation: Writing Scientific Software - Section 2.4

4. The solutions of the equation $x^2 + bx + c = 0$ can be calculated in two ways:

Algorithm 1	Algorithm 2
$D = \sqrt{\left(\frac{b}{2}\right)^2 - c}$	$D = \sqrt{\left(\frac{b}{2}\right)^2 - c}$
$x_1 = -\frac{b}{2} + D$	$x_1 = \text{sign}(-b)\ \left(\left\lvert\frac{b}{2}\right\rvert + D\right)$
$x_2 = -\frac{b}{2} - D$	$x_2 = \frac{c}{x_1}$

Download the program `quadratic.f90`, that implements both these algorithms in single-precision, from Toledo and have a look at the source code to see how it works. Now complete the following exercises:

- Add a double-precision version of each algorithm. **Make sure that your implementation is compiler independent.**
- In the main program, read b and c from the standard input stream and print the solutions of algorithms 1 and 2 both in single and double precision.
- Next, try some values for b and c. For example $b = 767$ and $c = 17$, $b \gg c$ or $b = c = 0$. Can you conclude which algorithm is numerically most stable?
- *Extra:* Explore the floating point exception handling possibilities from the F2003 standard. Write on the *standard error stream* (its unit number is given by the variable `error_unit` in the `iso_fortran_env` module) whether the *division-by-zero exception* and the *inexact-assignment exception* are supported by your CPU. Next, try to detect one of the issues encountered above using the corresponding error flags. For this part of the exercise, you will need the modules `ieee_exceptions` and `iso_fortran_env`.

Link preparation: Fortran 2003 standard - 14 (Exceptions and IEEE arithmetic)