Homework #3 — Numerical Limits:Overflow and Precision

Date: October 9, 2025
Student: Fariba Naghizadeh Avilagh

Objective

The purpose of this homework was to explore the numerical limits of integer and real data types in Fortran through two experiments:

1. **Integer overflow** — finding the point where factorial calculations exceed the representable range.
2. **Loss of significance** — finding when adding a very small real number to 1.0 no longer affects the result.

Task 1 — Integer Overflow

Two integer variables of kinds integer(kind=4) and integer(kind=8) were used to compute factorials sequentially.

- The 32-bit integer (kind=4) overflowed at **n = 13**, since 13! = 6,227,020,800 exceeds the maximum 32-bit limit ($\approx 2.1 \times 10^9$).
- The 64-bit integer (kind=8) overflowed at **n = 21**, since 21! = 51,090,942,171,709,440,000 exceeds the 64-bit range ($\approx 9.2 \times 10^{18}$).

When overflow occurred, the factorial value became negative or smaller than the previous one, confirming that the result could no longer be represented correctly.

Task 2 — Loss of Significance

Starting with big = 1.0d0 and small = 1.0d0, the program repeatedly halved small until big + small no longer changed the result.
The addition stopped affecting big after about **step 52**, indicating the **machine epsilon** for double precision ($\sim 2.22 \times 10^{-16}$).
This shows the finite precision of floating-point representation in Fortran.

Task 3 — Reflection

- **Integer limitation:** caused by a *finite range* — once the result exceeds the maximum storable value, it wraps around (overflow).
- **Real limitation:** caused by *finite precision* — small increments can no longer be represented when the gap between floating-point numbers exceeds the added value.

- **Most dangerous for simulations:** *Precision loss* is generally more problematic, as it can accumulate gradually and distort results without immediate detection, while overflow usually produces an obvious error.

Conclusion

This experiment demonstrated how numerical limits in Fortran arise from the binary representation of data types.
Integer overflow occurs when results exceed the range defined by the number of bits, while floating-point precision loss occurs when numbers become smaller than what can be resolved within the mantissa's bit capacity.

The bonus challenge reinforced these ideas:

- 13! and 21! revealed the exact overflow points for 32-bit and 64-bit integers, linking directly to their 31-bit and 63-bit signed ranges.
- Step 53 in the precision test revealed the machine epsilon of double precision ($\sim 2.22 \times 10^{-16}$); the final value was half of epsilon.

Understanding these numerical limits is essential for writing reliable scientific code, as it helps prevent overflow, detect precision loss, and ensure the stability and accuracy of numerical simulations.