Course logistics

Verification and validation

Representation of floating point numbers
Overview of 6810 Computational Physics

- Each session: brief overview/recap then work through “session guide” (e.g. 6810: 1094 Activities 1) with a partner
  - Night before class: read session notes (≈ 10 pages)
  - Pick up new or returned activities guides
  - Each student has own computer (but talk to each other!)
  - Fill in answers and hand in at end (unless told otherwise)
  - “Activities” will often take more than one period to complete
Overview of 6810 Computational Physics

- Each session: brief overview/recap then work through “session guide” (e.g. 6810: 1094 Activities 1) with a partner
  - Night before class: read session notes (≈ 10 pages)
  - Pick up new or returned activities guides
  - Each student has own computer (but talk to each other!)
  - Fill in answers and hand in at end (unless told otherwise)
  - “Activities” will often take more than one period to complete

- Course info (Goggle “6810 OSU” to get to web pages)
  - Look at main web page (readings and downloads), info page (references and other course info), gameplan page
  - Instructors: Dick Furnstahl, Bryan Smith
  - Office hours: hope to use Sm1094 (stay tuned!)
  - Grade based on effort and (scaled) accomplishment
    - session sheets graded ✓, +, −; always upgradable
    - homework assignments (follow-ups to class); usually ≈ 4
    - independent project (more later!)
  - Carmen (Canvas) course page for scores

Dick Furnstahl 6810: Session 1
Overview of 6810 Computational Physics

- Each session: brief overview/recap then work through “session guide” (e.g. 6810: 1094 Activities 1) with a partner
  - Night before class: read session notes (≈ 10 pages)
  - Pick up new or returned activities guides
  - Each student has own computer (but talk to each other!)
  - Fill in answers and hand in at end (unless told otherwise)
  - “Activities” will often take more than one period to complete

- Course info (Google “6810 OSU” to get to web pages)
  - Look at main web page (readings and downloads), info page (references and other course info), gameplan page
  - Instructors: Dick Furnstahl, Bryan Smith
  - Office hours: hope to use Sm1094 (stay tuned!)
  - Grade based on effort and (scaled) accomplishment
    - session sheets graded ✓, +, −; always upgradable
    - homework assignments (follow-ups to class); usually ≈ 4
    - independent project (more later!)
  - Carmen (Canvas) course page for scores

- Questions?
Pedagogical philosophy

- See Session 1 notes and 6810 gameplan webpage for details
- **Approach:** “Throw you in the water to learn how to swim”
- Like physics research: YOU have to figure things out (which means you will be confused much of the time)
- **Spiral approach:** keep returning to learn things incrementally
- “Just enough” detail is given (plus clues)
- **Quicker to modify programs** (trick is to get you to read code!)
  - identify mistakes (“bugs”) that need to be fixed
  - changes that must be made to accomplish task
- **Try to predict but always postdict. Question authority!**
- Ask about *anything* (connected to physics/computing :)

Dick Furnstahl

6810: Session 1
Why GNU/Linux, command-line, C++, Python, ... ?

- Physics approach: look inside black box
- Large-scale computation uses Fortran 2008 or C++
- Python as *scripting* language (although gaining for HPC)
- Easy to segue to IDE’s later (e.g., Eclipse)
- 99% of Top 500 Supercomputers (and most clusters) use Linux
Session 1 Plan: Basics we’ll need this semester

- Get started with Unix command line environment on Linux or Windows/Cygwin
- Successfully download compile, run, and modify a “Hello, World!” C++ program (and some other simple programs)
- Try out `make` and use g++ warnings
- Understand how underflow, overflow, and machine precision limits can be determined “experimentally”
- Try out a program that uses a special function from the Gnu Scientific Library (GSL)
- Try out Python
- Activate BuckeyeBox account. We’ll use it to hand in homework.
- Setting up your own machine: details to follow (but ask).
Verification and validation \(\Rightarrow\) Checks

- Definitions: Validation ensures that “you built the right thing”. Verification ensures that “you built it right”.
- Basic questions to always ask: “How do you know a program is working correctly? And to the accuracy it should?"
- Getting an answer (no error messages) is not enough!
- Think about how to check your answer (be creative!)
  - units
  - analytic (“model”) solution to compare (usually simpler case)
  - solve by another method
  - special or limiting cases
  - scaling laws \[ T = 2\pi \sqrt{L/g} \Rightarrow T \text{ should double if } L \rightarrow 4L \]
- Programming [see Session 1 notes and Hjorth-Jensen 1–2]
  - Use compiler to find errors (turn on all warnings)
  - Good habits \(\Rightarrow\) reliability, portability, extensibility
  - E.g., pseudocodes, comments, formatting, variable name choice, ...
Computational math ≠ regular math

- Two main issues:
  - approximate representation of floating point ("with decimal points") numbers → only finite # of *machine numbers*
  - binary (base 2) representation

- Think about base 10 first ("Do the simple case first")
  - normalized scientific notation, e.g., 35.2163 → $0.352163 \times 10^2$
  - general: $x = \pm r \times 10^n$ with $1/10 \leq r < 1$

- store sign, mantissa, exponent
- suppose we store sign, 6-digit mantissa, 1-digit exponent

\[
-\frac{4}{3} = -1.333\overline{3} \approx (-1) \times (0.133333) \times 10^1
\]

\[
= (-1) \times (0.133333) \times 10^{[6-5]}
\]

with "bias" so we get + or − exponents without storing sign

- Only a finite # of numbers → largest, smallest
Computational math ≠ regular math

- Continue with decimal example: sign, 6-digit mantissa, 1-digit exponent with bias

- Largest: $0.999999 \times 10^{[9-5]} = 9999.99$ (so $\geq 10000$ will overflow)
- Smallest $0.100000 \times 10^{[0-5]} = 0.000001$ (so $10^{-7}$ will underflow)

- Most numbers not represented exactly (cf. machine numbers)
  - $z_c = z(1 + \epsilon)$ where $-\epsilon_m < \epsilon < \epsilon_m$
  - for our decimal example, $\epsilon_m \approx 10^{-6}$
  - $\epsilon_m$ is “machine precision” $\Rightarrow$ worst case of gap

- Consequences of finite precision: no limits! (no $\epsilon \rightarrow 0$)
  - $x = 3500 = 0.35 \times 10^{[9-5]}$ and $y = 0.0021 = 0.21 \times 10^{[2-5]}$
    both represented accurately
  - but $x + y = x$ (!) $\Rightarrow$ round-off error!
  - subtraction of similar numbers is even worse

- What about derivatives, integrals, etc.?
Decimal vs. binary

- Recall how numbers are represented in decimal (base 10):
  \[
  \begin{array}{ccccccccc}
  10^3 & 10^2 & 10^1 & 10^0 & 10^{-1} & 10^{-2} & 10^{-3} \\
  10 & 3 & 1 & 2 & 4 & . & 1 & 2 & 3 \\
  \end{array}
  \]

- Analog in binary (base 2):
  \[
  \begin{array}{ccccccccc}
  2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\
  1 & 0 & 1 & 1 & . & 0 & 1 & 1 \\
  \end{array}
  \]

- Finite decimals may not be binary machine numbers!
  - machine numbers: $5 \rightarrow 101$, $1/4 \rightarrow 0.01$
  - but what about $1/5$ or $1/3$? [answers: $0.0011$ and $0.01$]
  - binary long division:
    \[
    \begin{array}{c|cccc}
    & 1 & 0 & 1 & 1 \\
    \hline
    3 & 1 & & & \\
    \hline
    11 & 1 & & & \\
    \end{array}
    \]

- What are the largest, smallest binary numbers? What is $\epsilon_m$? (warning: IEEE standard for binary representation differs!)