WS 25/26 Numerik 0 Notes

Igor Dimitrov

2025-10-10

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Intro

1.1 Repo structure and Usage

1.1.1 Repo Layout

```
num-sol-ws2526/
  hdnum/                # submodule
  CMakeLists.txt        # builds all src/ubN targets
  src/
     ub1/
        CMakeLists.txt
        ub1_task1.cpp
     ub2/
         CMakeLists.txt
  theory/
     ub1/
     ub2/
  .github/workflows/build.yml
```

1.1.2 Teammate onboarding snippet

Send them this:

```
# First-time clone (includes submodule)
git clone --recurse-submodules https://github.com/igor-dimi/num-sol-ws2526.git
cd num-sol-ws2526

# Build
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j

# Create your branch for this week
git checkout -b ub1_<yourname>
# ...work...
```

```
git commit -am "ub1: your message"
git push -u origin ub1_<yourname>
```

1.1.3 Day-to-day workflow (lightweight)

- **Sync main (everyone):**

  ```
  git checkout main
  git pull --ff-only
  git submodule update --init --recursive
  ```

- **Work on a branch for the week:**

  ```
  git checkout -b ub1_igor
  # edit src/ub1/... and theory/ub1/...
  cmake -S . -B build -DCMAKE_BUILD_TYPE=Release   # usually only first time
  cmake --build build -j                           # build locally
  git add -A
  git commit -m "ub1: implement task1; add notes"
  git push -u origin ub1_igor
  ```

- **Merge strategy:**
  - If your changes don't conflict and are small: either **open a PR and self-merge** after CI is green **or** fast-forward/merge to `main` directly.
  - If you both touched the same files or it's risky: **open a PR** and ask for a quick review.

  With this setup we **don't** enable branch protection. CI will still run and show you if `main` breaks, but it won't block merges.

1.1.4 Optional niceties (quick wins)

- **Labels**: create `ub1`, `ub2`, … labels; tag issues/PRs by sheet.
- **CODEOWNERS** (optional): `.github/CODEOWNERS`

  ```
  /src/ub1/  @igor-dimi @teammate
  /src/ub2/  @igor-dimi @teammate
  ```

  (This just auto-requests reviews; it won't block merges unless you add protection later.)
- **CI variants**: if you don't want GMP on CI, drop `libgmp-dev` and keep `HDNUM_USE_GMP=OFF` as default in CMake.

1.1.5 Example Workflow

Two options:

- **linear (rebase + fast-forward)** way first (preferred),
- simple **merge commits** way.

A) Linear history (rebase each branch onto `main`, then fast-forward)

This avoids “merge bubbles” and keeps history tidy.

1) Merge your branch

```
#### make sure main is up to date
git checkout main
git pull --ff-only

#### rebase your branch on top of the latest main
git checkout ub1_igor
git fetch origin
git rebase origin/main        # resolve conflicts if any: edit -> git add -> git rebase --co

#### sanity check
cmake --build build -j || { echo "Fix build before merging"; exit 1; }

#### fast-forward main to include your branch
git checkout main
git merge --ff-only ub1_igor  # will fail if rebase wasn't done; that's good
git push
```

2) Merge your teammate’s branch

```
git checkout ub1_malte
git fetch origin
git rebase origin/main        # now 'main' already contains your work
#### resolve conflicts if any -> git add -> git rebase --continue
```

```
cmake --build build -j

git checkout main
git merge --ff-only ub1_malte
git push
```

If during the rebase you had already pushed your branch earlier, you'll need to update it (optional) with:

`git push --force-with-lease`

Use `--force-with-lease` (not plain `--force`) to avoid clobbering a teammate's work accidentally.

B) Simple merge commits (no rebases)

This is quickest if you don't care about a perfectly linear history.

```
git checkout main
git pull --ff-only

#### merge your branch
git merge --no-ff ub1_igor    # creates a merge commit even if FF would be possible
#### or: git merge ub1_igor      # lets Git fast-forward if possible
cmake --build build -j
git push

#### merge your teammate's branch
git merge --no-ff ub1_malte
#### resolve conflicts if prompted: edit -> git add <files> -> git commit
cmake --build build -j
git push
```

Conflict handling (both methods)

- Git stops and shows `CONFLICT` markers if you happened to touch the same lines.
- Open the files, keep the correct pieces, then:

  ```
  git add <fixed-files>
  # continue the operation:
  # - during rebase: git rebase --continue
  # - during merge:  git commit
  ```

- Rebuild locally; only push once it compiles.

Submodule note

If neither branch changed the `hdnum` submodule pointer, nothing special. If one did, after merging:

```
git submodule update --init --recursive
```

Which to choose?

- **A) Rebase + `--ff-only`**: best if you value a clean, straight history.
- **B) Merge commits**: fine for a small class repo; fewer commands; history will have merge nodes.

1.2 CMAKE

1.2.1 2) Top-level CMake for examples (`code_examples/my_solutions/CMakeLists.txt`)

- Defines a shared **INTERFACE** target `hdnum_common` (provides include paths, optional GMP).
- Adds each chapter as a subdirectory if it exists.

```
cmake_minimum_required(VERSION 3.16)
project(numerics_solutions CXX)
set(CMAKE_CXX_STANDARD 20)

# Put binaries under build/bin/ubX/
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${CMAKE_BINARY_DIR}/bin")

# hdnum include dir (submodule at repo root)
set(HDNUM_DIR "../hdnum")

# Shared interface target with include path (and optional GMP)
add_library(hdnum_common INTERFACE)
target_include_directories(hdnum_common INTERFACE "${HDNUM_DIR}")

option(HDNUM_USE_GMP "Enable GMP in hdnum" OFF)
if(HDNUM_USE_GMP)
  target_compile_definitions(hdnum_common INTERFACE HDNUM_HAS_GMP=1)
  find_library(GMPXX gmpxx)
  find_library(GMP   gmp)
  if(GMPXX AND GMP)
    target_link_libraries(hdnum_common INTERFACE ${GMPXX} ${GMP})
  else()
    message(FATAL_ERROR "GMP not found; install libgmp-dev or disable HDNUM_USE_GMP")
  endif()
endif()

# Add each UB subdir in src/ if it has a CMakeLists.txt
foreach(ub IN ITEMS ub1 ub2 ub3 ub4 ub5 ub6 ub7 ub8 ub9 ub10)
  if(EXISTS "${CMAKE_CURRENT_LIST_DIR}/${ub}/CMakeLists.txt")
    add_subdirectory("${ub}")
  endif()
endforeach()
```

1.2.2 3) Per UB-folder CMake with per-Ub output folders

- Each chapter decides which executables to build.
- **Per-chapter runtime output** goes to `build/bin/<chapter>/` (and per-config subfolders on multi-config generators).

```
set(UB ub1)
```

```
# Group executables per UB on disk (and per-config for multi-config generators)
set(OUT "${CMAKE_BINARY_DIR}/bin/${UB}")
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${OUT}")
foreach(cfg IN ITEMS Debug Release RelWithDebInfo MinSizeRel)
  set(CMAKE_RUNTIME_OUTPUT_DIRECTORY_${cfg} "${OUT}/${cfg}")
endforeach()

function(add_ub_example name)
  add_executable(${name} "${name}.cpp")
  target_link_libraries(${name} PRIVATE hdnum_common)
  set_target_properties(${name} PROPERTIES FOLDER "${UB}") # IDE grouping
endfunction()

# List ub1 binaries here:
add_ub_example(ub1_task1)
add_ub_example(ub1_task2)
```

Repeat a similar `CMakeLists.txt` in `src/ub02`, `src/ueb03`, … adding that chapter's `.cpp` files.

1.2.3 4) Build commands (configure once, then build)

Single-config (Linux/Mint, Makefiles/Ninja):

```
cd src
### configure (once per build dir or when options/CMakeLists change)
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release                # + -DHDNUM_USE_GMP=ON if neede
### build (repeat as you edit sources)
cmake --build build -j
```

Outputs

```
build/bin/ub01/ub01_ode_demo
build/bin/ub01/ub01_newton_demo
build/bin/ub02/...
```

1.2.4 5) Using GMP (high precision) without editing submodule

- Install dev package: `sudo apt install -y libgmp-dev`

- Enable once at configure time (persists in the build cache):

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release -DHDNUM_USE_GMP=ON
```

- CMake finds and links `gmpxx/gmp`, and defines `HDNUM_HAS_GMP=1` for all examples via `hdnum_common`.

  Tip: keep **two build dirs** if you switch often:

```
cmake -S . -B build         -DCMAKE_BUILD_TYPE=Release -DHDNUM_USE_GMP=OFF
cmake -S . -B build-gmp     -DCMAKE_BUILD_TYPE=Release -DHDNUM_USE_GMP=ON
cmake --build build
cmake --build build-gmp
```

1.2.5 6) Troubleshooting quickies

- `fatal error: hdnum.hh: No such file or directory` → `HDNUM_DIR` wrong; ensure submodule is initialized; include path points at the folder that **contains** `hdnum.hh`.
- `GMP not found` → install `libgmp-dev`; reconfigure; or pass custom `-DCMAKE_LIBRARY_PATH=/path -DCMAKE_INCLUDE_PATH=/path`.
- Changing `HDNUM_USE_GMP` or editing `CMakeLists.txt` → reconfigure (rerun the `cmake -S . -B build ...` step). Otherwise just `cmake --build build -j`.