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/
  CMakeLists.txt                  # tiny forwarder: just add_subdirectory(src)
  hdnum/                          # submodule (header-only HDNUM library)
  src/                            # build root for all programming exercises
     CMakeLists.txt               # real top-level: sets hdnum include, adds all ubN subdirs
     ub1/
        CMakeLists.txt            # defines ub1 targets (e.g., ub1_task1), sets bin dir to bu
        ub1_task1.cpp
     ub2/
        CMakeLists.txt            # same pattern for week 2 (easy place to add per-UB flags)
        ub2_task1.cpp
     ...
  theory/
     ub1/
     ub2/
  .github/workflows/
     build.yml                    # CI: configure with -S . (root) now that we have a forward
  .gitignore
```

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 and Git Guide

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 # from the src folder

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 ub01 ub02 ub03 ub04 ub05 ub06 ub07 ub08 ub09 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):

```
### configure (once per build dir or when options/CMakeLists change)
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release              # + -DHDNUM_USE_GMP=ON if needed
### 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`.

1.3 Commit Guide

here's a single, self-contained guide you can drop into your notes. it pulls together: clear commit message style, when/where to commit, fixup commits, squashing (locally and via PR), CI considerations, and safety tips.

1.3.1 why commits matter (even in a small class repo)

clean commits make it easy for teammates (and future you) to understand what changed and why. your history is your lab notebook: compact, accurate, readable.

1.3.2 guiding principles

- commit **often while you work**, but **present a tidy history when you merge**.
- keep `master`/`main` always **working**; do experimental work on branches like `ub1_igor`.
- use CI to sanity-check builds, not to gate every tiny doc tweak.
- write commit messages that explain **why**, not only **what**.

1.3.3 commit message style

short template

```
<type>(<scope>): <short, imperative summary>

Why:
- a brief explanation of motivation/context
- any side effects, constraints, or follow-ups
```

Common types: `feat`, `fix`, `docs`, `ci`, `build`, `refactor`, `chore`, `merge`.

examples tailored to your repo

```
feat(build): add root CMakeLists forwarder to src/

Why:
- allows 'cmake -S . -B build' at repo root
- improves IDE support that expects a top-level CMakeLists

feat(build): per-UB output dirs under build/bin/ubN/

Why:
- keeps binaries organized by sheet
- avoids mixing artifacts across weeks

feat(build): add hdnum_common interface target

Why:
- centralizes include path and optional GMP flags
- reduces repetition across ubN CMakeLists

fix(build): CI configure uses -S . with root forwarder

Why:
- matches local build instructions after adding root CMakeLists

feat(build): add HDNUM_USE_GMP option (default OFF)

Why:
- enable high precision when libgmp-dev is available
- keep default lightweight for CI and new clones
```

```
feat(ub1): add ub1_task1 using hdnum::Vector

- minimal example: fill vectors, combine, manual dot product
- adds per-UB CMakeLists template

feat(ub2): Newton demo for sqrt(a) with analytic Jacobian

Why:
- aligns with sheet 2 theory
- demonstrates hdnum::Newton usage and stopping criteria

refactor(ub1): extract print_vec helper; tidy includes

Why:
- remove duplication across ub1 tasks

fix(ub1): correct A*x multiplication order in minimal example

- y[0] now matches expected result

docs(ub1): add notes.qmd skeleton; link to code listings

- file-backed blocks include src/ub1/*.cpp

docs(readme): clarify -S . vs -S src with forwarder example

- add decision tree and CI note

chore(submodule): add hdnum as submodule at repo root

- pinned to commit abc1234
- usage: git submodule update --init --recursive

chore(submodule): bump hdnum to upstream vX.Y (abcdef0)

Why:
- picks up bugfix in ExplicitEuler step control

ci: add build workflow on push/PR; fetch submodules

- installs cmake, g++, libgmp-dev
- configures with -S .
```

```
ci: trigger only on src/** and workflows to reduce noise

Why:
- avoid rebuilding on docs-only changes


ci: cancel superseded runs for same branch

concurrency:
  group: build-${{ github.ref }}
  cancel-in-progress: true


feat(tools): Makefile to copy qmd+sources to Seafile (NR param)

- rsync only newer files for theory/src


feat(share): Makefile for incremental render+zip in ubN/

- renders only stale PDFs
- zips PDFs and sources as Heinrich_BlattNN_*.zip


chore(branch): rename default branch master -> main

- update CI triggers to [ main, master ] for transition


merge: squash ub1_igor into master (UB1 solutions)

- ExplicitEuler example
- notes.qmd
- per-UB CMakeLists
```

1.3.4 where to commit: branches vs master

- **on your UB branch** (`ub1_igor`, `ub1_malte`): commit as often as you like; small steps are fine.
- **on master**: only push **working**, self-contained changes (build locally first). prefer merging/squashing from a branch over committing directly.

for tiny docs like `README.md`, direct commits to master are fine—especially if your CI is configured to trigger only on `src/**`.

1.3.5 making small corrections without clutter

amend the last commit (not pushed yet)

```
git add -A
git commit --amend
```

updates the previous commit in place (no new commit).

fixup commits (already pushed or want to mark follow-ups)

create small "fixup" commits that will be auto-squashed later:

```
git commit --fixup <target-commit-sha>
### later:
git rebase -i --autosquash origin/master
```

configure autosquash by default:

```
git config --global rebase.autosquash true
```

1.3.6 squashing commits (make history tidy)

fast one-liner: squash last N commits into one

```
git reset --soft HEAD~N
git commit -m "ub1: final solution and notes"
```

use before pushing, or be ready to force-update your branch.

interactive rebase (full control)

```
git rebase -i HEAD~N
```

change the list to `squash`/`fixup` the follow-ups into the first commit, edit the final message, then:

```
git push --force-with-lease
```

(`--force-with-lease` prevents clobbering a teammate's remote work.)

squash when merging (GitHub UI)

open a PR and click **"Squash and merge"**. github creates **one** commit on master containing all your branch changes, regardless of intermediate commits.

squash without a PR (local)

```
git checkout master
git merge --squash ub1_igor
git commit -m "ub1: solutions"
git push
```

1.3.7 merging two UB branches that touched different files

option A (clean & linear):

```
git checkout master
git pull --ff-only

git checkout ub1_igor
git rebase origin/master
cmake --build build -j
git checkout master
git merge --ff-only ub1_igor
git push

git checkout ub1_malte
git rebase origin/master
cmake --build build -j
git checkout master
git merge --ff-only ub1_malte
git push
```

option B (quicker, allows merge commits):

```
git checkout master
git pull --ff-only
git merge ub1_igor
cmake --build build -j
git push
```

```
git merge ub1_malte
cmake --build build -j
git push
```

1.3.8 CI etiquette (so you don't pay for needless runs)

- trigger builds only when relevant files change:

```
on:
  push:
    branches: [ master ]
    paths:
      - 'src/**'
      - '.github/workflows/**'
  pull_request:
    branches: [ master ]
    paths:
      - 'src/**'
      - '.github/workflows/**'
```

- optionally cancel superseded runs if you push multiple times quickly:

```
concurrency:
  group: build-${{ github.ref }}
  cancel-in-progress: true
```

- for docs-only commits, you can use `[skip ci]` in the message (sparingly).

1.3.9 when to open a PR vs pushing to master

- open a **PR** when:
  - both of you touched the same files
  - the change is risky or large
  - you want a quick review (PR = free proofreading + CI view)
- push or fast-forward merge to **master** when:
  - small, low-risk, and you built locally
  - docs-only changes

1.3.10 submodule gotcha refresher (hdnum)

- after pulling, always align the submodule to the recorded commit:

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

- don't edit `hdnum/` in place; if you must, fork and repoint the submodule.
- enabling GMP is a **build flag**, not an edit to hdnum:

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

1.3.11 safety net: undo & recover

- view recent HEADs:

```
git reflog
```

- hard reset to a prior state (careful; this rewrites your working tree):

```
git reset --hard <reflog-hash-or-commit-sha>
```

- abort an in-progress rebase or merge:

```
git rebase --abort
git merge --abort
```

1.3.12 small "before you merge to master" checklist

```
### on your UB branch
git fetch origin
git rebase origin/master        # optional but tidy
cmake --build build -j          # local build passes
git log --oneline origin/master..HEAD  # commits look meaningful
### decide: squash locally (rebase -i) or Squash-and-merge via PR
```

1.3.13 tl;dr

- commit freely on branches; **squash** before merging.
- keep master **working**; build locally first; let CI sanity-check.
- write messages that say **why**.
- use `--fixup` + autosquash for painless cleanup.
- never rewrite shared branch history; **do** rewrite your feature branch history (then `--force-with-lease`).