

Beadandó a Kutatómunka információs eszközei nevű tárgyhoz

Kormányos Hanna Rebeka
P2ONMD

2019.05.26.



Tartalomjegyzék

1. Bevezető	3
2. A feladatok	3
2.1. 2x2 mátrix struktúra készítése	3
2.2. NxN-es mátrix osztály készítése	3
3. A használt eszközök bemutatása	3
3.1. C++ használata	4
3.2. VSCode	5
3.2.1. Futtatási módok	5
3.3. GitHub	6
3.4. Új repozitórium létrehozása	6
3.5. Új verzió feltöltése és dokumentálása	7
3.6. Cmake Pipeline	7
3.6.1. Időmérés	7
3.6.2. Pipeline	7

1. Bevezető

Ebben a beadandóban a Haladó alkalmazott programozás nevű tárgyra készített két feladat elkészítése közben használt eszközöket, problémákat és módszerek fogom bemutatni.

2. A feladatok

2.1. 2x2 mátrix struktúra készítése

A feladat 2x2 mátrix struktúra készítése a következő műveletekkel:
Beépített és külső formában is:

- összeadás
- kivonás
- skalárral szorzás (két oldalról)
- skalárral osztás

További műveletek:

- mátrix vektorral való szorzása (A feladatban megkülönböztetjük a sor és oszlop vektorokat.)
- determináns számolás
- transzponálás
- invertálás
- egyenletrendszer megoldás (Cramer szabállyal)

2.2. NxN-es mátrix osztály készítése

A feladat egy NxN-es (négyzetes) mátrix osztályt készítése, ami `std::vector`-ban tárolja az elemeket, lehet indexelni és megvannak rá írva az alapvető műveletek (a fenti feladathoz hasonlóan) és ezeknek az operátoroknak az összes `&&`-es változatai is.

Emellett a feladat része még a mátrix objektumunkból kettőt véletlen számokkal feltölteni és lemérni, hogy mennyi idő összeszorozni őket különböző N-ek esetén (N^3 -ös skálázást várunk).

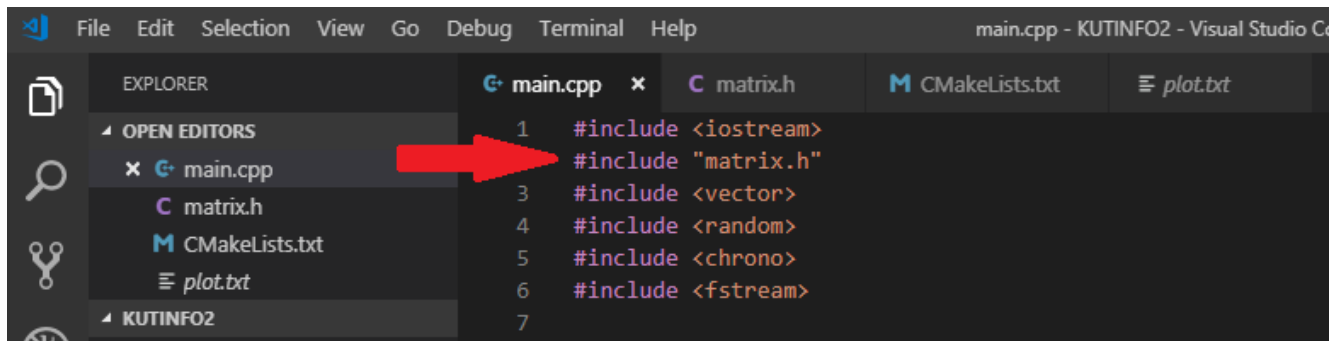
3. A használt eszközök bemutatása

- C++17- standard library (programozási nyelv)
- CMake 3.0.0 (fordító)
- Microsoft Visual Studio Code (VSCode) (fejlesztői környezet)

- GitHub (verziókövető kódrepozitóriumban)
- Gnuplot (függvényrajzoló program)
- LaTeX (szövegformázó program)

3.1. C++ használata

Az NxN-es mátrix osztály megírásához az áttekinthetőség érdekében külön headert készítettem, ami magát a struktúrát és az azon kívüli műveleteket tartalmazza. Ezt a headert includolni kell ezután a main.cpp-be ahogy az alábbi ábra mutatja.



1. ábra. Külön header használata, includeok

A programok megírásához a következő includeokat használtam a standard library által nyújtott lehetőségeken kívül:

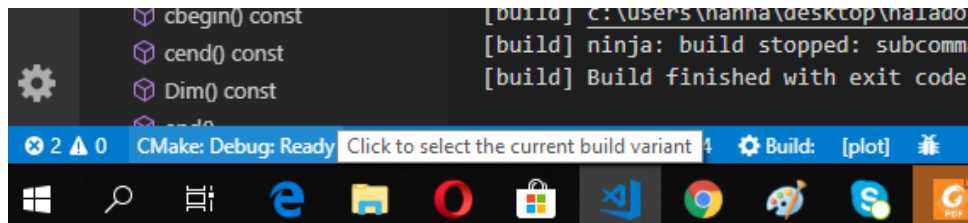
- `<iostream>` (preprocesszor direktíva)
- `<cmath>` (beépített matematikai függvények)
- `<array>` (std::array betöltése)
- `<vector>` (std::vector betöltése)
- `<ostream>` (Streamek kimeneti fájlba írásához)
- `<fstream>` (Input/output streameket kezelő csomag)
- `<initializer_list>` (T típusú típusú objektumok kezelése)
- `<algorithm>` (elemtartományokra használandó funkcióka)
- `<random>` (random számok generálása)
- `<chrono>` (időmérés)

3.2. VSCode

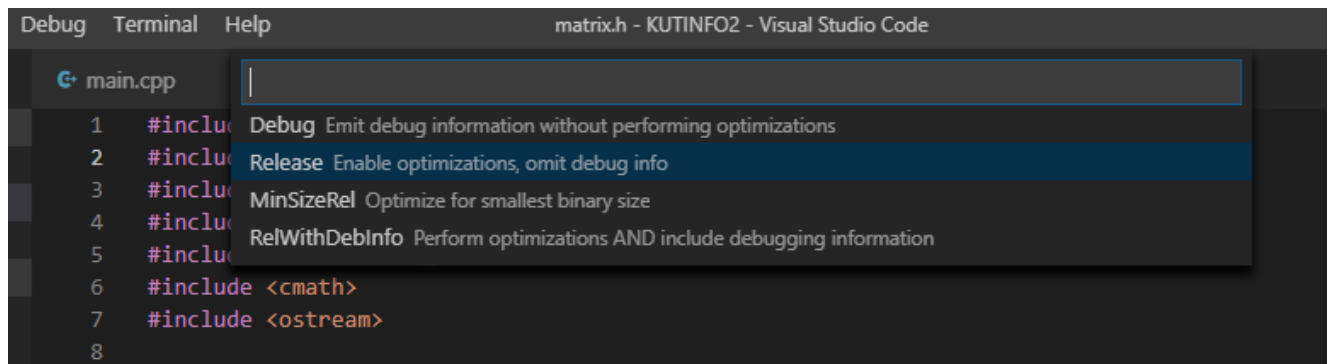
A Visual Studio Code egy forráskódszerkesztő, amely számos programozási nyelven használható. A Microsoft fejlesztette ki a Windows, Linux és MacOS számára. Magában foglalja a hibakeresés, a beágyazott Git vezérlés és a GitHub támogatását, a szintaxis kiemelését, az intelligens kód befejezését, a töredékeket és a kódfrissítést.

3.2.1. Futtatási módok

Az NxN es mátrix osztály kódolása közben a Debug futtatási módot használtam. Ez úgy működik, hogy miközben futtatja a kódot keresi a hibákat is szóval ez egy lassabb folyamat. Viszont amikor már élesben ment az időmérés a Release verziót használtam, ekkor Debugolással már nem foglalkozik, úgy rendezi át a kódot a fordító, hogy a futási idő a leghatékonyabb legyen.



2. ábra. Futtatási módok



3. ábra. Futtatási módok kiválasztása

3.3. GitHub


A feladat megoldás során szerettünk volna egy verziókövető rendszert használni. Verziókezelés alatt több verzióval rendelkező adatok kezelését értjük. Ez azért nagyon hasznos, mert áttekinthetjük a fejlesztési folyamatot, valamint ha a változtatások során hibát követünk el könnyen visszaállítható az eredeti kód. A verziókezelés másik fontos tulajdonsága hogy a projekteken való változtatásokat egyszerűen tudjuk könyvelni. A github.com verziókövető weboldalt használtuk az órán, létrehoztunk egy profilt, majd egy Git repozitóriumot. A rendszer jól működött, nagyon hasznosnak bizonyult.

3.4. Új repozitórium létrehozása


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

 korhanreb ▾

Repository name *

/ Kutinfo 

Great repository names are short and memorable. Need inspiration? How about **psychic-potato**?

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.



Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

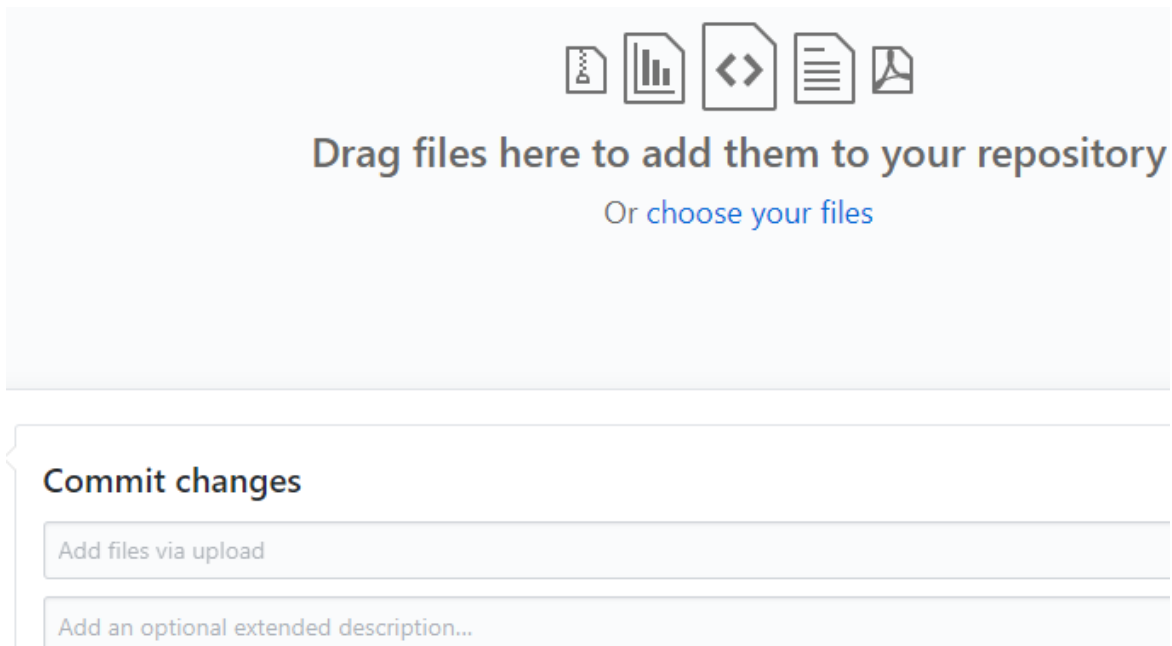
Add a license: None ▾



Create repository

4. ábra. Új repozitórium

3.5. Új verzió feltöltése és dokumentálása



5. ábra. Feltöltés és dokumentálás

3.6. Cmake Pipeline

3.6.1. Időmérés

A Cmake ezen funkcióját a időfüggés ábrázolására használtam. Az időmérés, mint minden más esetben itt is csak egy felsőbecslét ad, hiszen a háttérben mindig fut valami más is, terheli a kapacitást, ezt nem tudjuk kiküszöbölni.

3.6.2. Pipeline

A CMake funkciója az ún. Pipeline. Ezáltal a fordítás során egy fájlba írt parancssorozatot tud a felhasználó egy külső programnak átadni, így közvetlenül a kapott eredmények kiértékelhetők, pl. Gnuplotban ábrázolhatók.

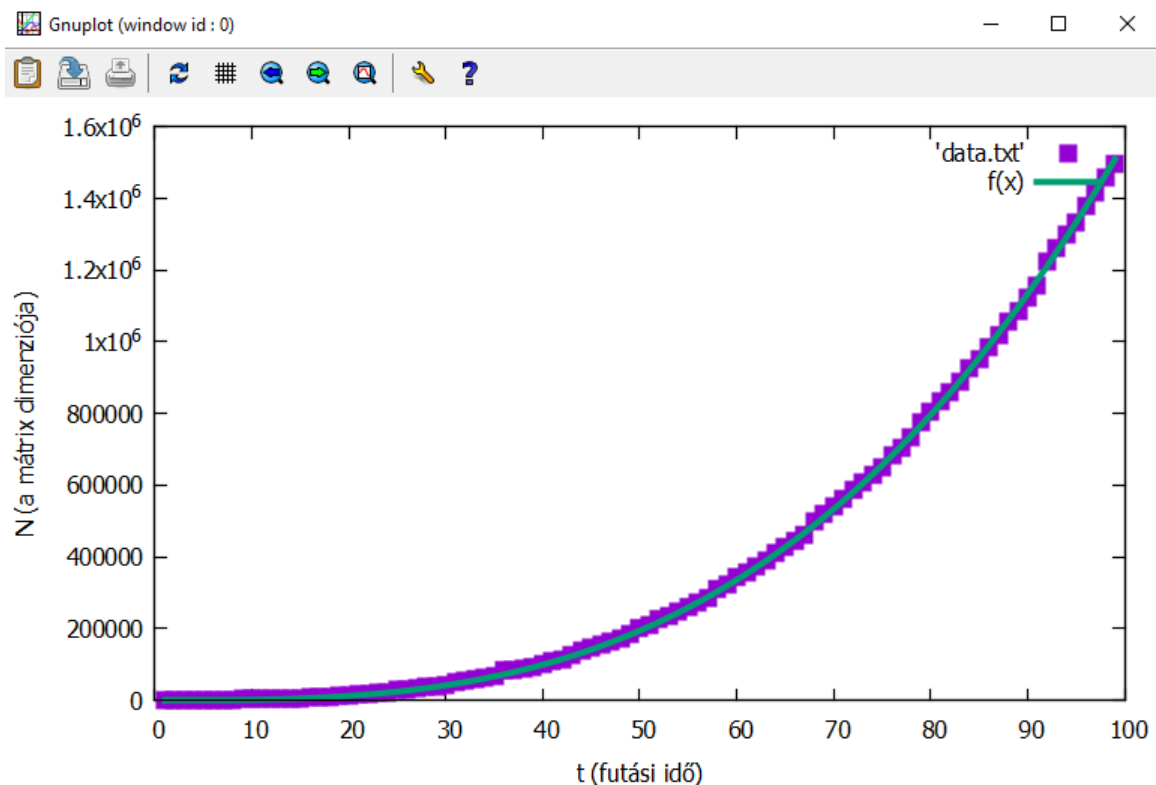
```

7  add_executable (idomeres main.cpp)
8  add_custom_command( COMMAND idomeres
9  WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
10 OUTPUT ${CMAKE_BINARY_DIR}/data.txt
11 DEPENDS idomeres
12 COMMENT "Generating data set")
13 add_custom_target(data ALL DEPENDS ${CMAKE_BINARY_DIR}/data.txt)
14
15 set(GNUPLOT_EXECUTABLE "C:\\Program Files\\gnuplot\\bin\\gnuplot.exe")
16
17 find_package (Gnuplot REQUIRED)
18
19 add_custom_command( COMMAND ${GNUPLOT_EXECUTABLE} ${PROJECT_SOURCE_DIR}/plot.txt
20 WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
21 OUTPUT ${CMAKE_BINARY_DIR}/data.png
22 DEPENDS ${PROJECT_SOURCE_DIR}/plot.txt data
23 COMMENT "Generating plot")
24 add_custom_target(plot ALL DEPENDS ${CMAKE_BINARY_DIR}/data.png)

```

6. ábra. Gnuplot használata Cmake Pipeline segítségével

Annak érdekében használtam tehát a Cmake Pipelineot, hogy a futtatással egyidőben rögtön lássam az időmérés eredményét, és újra tudjam futtatni, hogyha esetleg valami probléma történt. Például túl nagy a zaj az időfüggésben.



7. ábra. Futási idő- mátrix dimenzió grafikon készítése Gnuplottal


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
[build] iter      chisq      delta/lim  lambda  a
[build]
[build] After 3 iterations the fit converged.
[build] final sum of squares of residuals : 1.20561e+013
[build] rel. change during last iteration : -4.62752e-008
[build]
[build] degrees of freedom      (FIT_NDF)                : 98
[build] rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf) : 350744
[build] variance of residuals (reduced chisquare) = WSSR/ndf   : 1.23021e+011
[build]
[build] Final set of parameters          Asymptotic Standard Error
[build] =====
[build] a              = 200.142          +/- 0.09445      (0.04719%)
[build] Build finished with exit code 0

Studio Build Tools 2017 - amd64  Build: [plot]  Run CTest  main(int
```

8. ábra. Illesztett paraméter

Hivatkozások

- [1] https://en.wikipedia.org/wiki/Visual_Studio_Code
- [2] <https://en.wikipedia.org/wiki/Git>
- [3] <https://hu.wikipedia.org/wiki/Verziókezelés>