Eötvös Loránd Tudományegyetem

Informatikai Kar

Numerikus Analízis Tanszék

Hőtérkép készítése radiális bázisfüggvények segítségével

Témavezető:
Dr. Lócsi Levente
egyetemi adjunktus

Szerző:
Horváth Milán
programtervező informatikus BSc

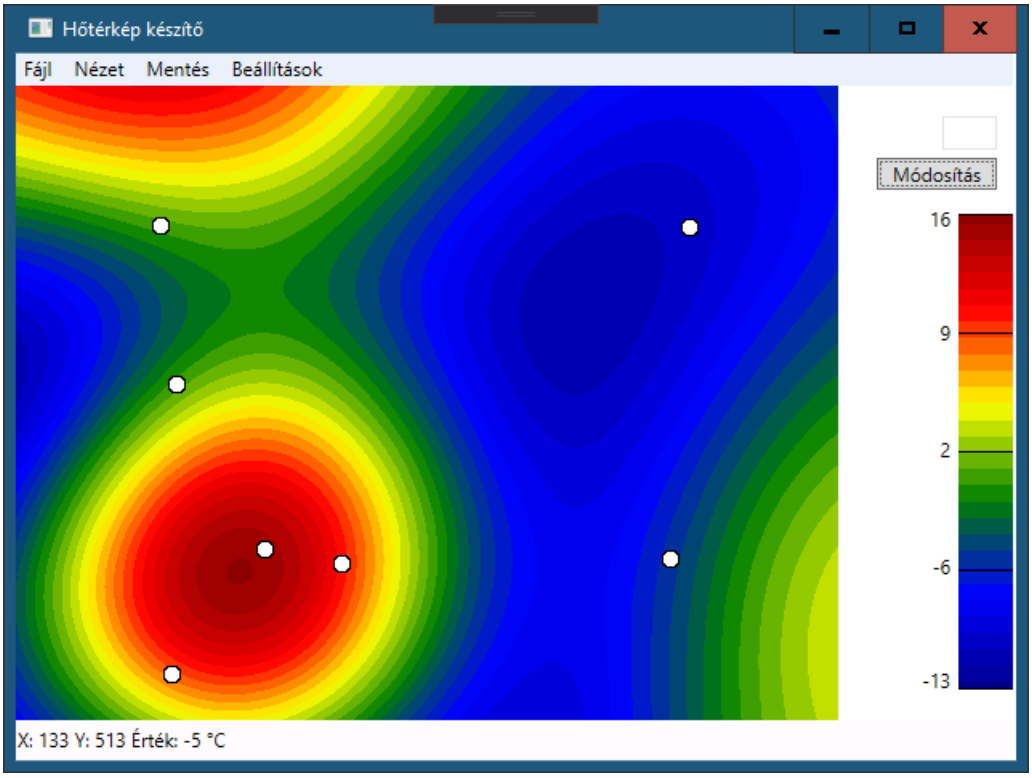
Budapest, 2020

Tartalomjegyzék

1. fejezet

Bevezetés

Szakdolgozatom alapjául a radiális bázisfüggvényekkel való közelítés részletes, és érthető bemutatását választottam. Ehhez célom könnyen kezelhető program létrehozása, mellyel bármely felhasználó könnyedén kipróbálhatja saját adatokkal a függvényközelítést. Igyekeztem törekedni olyan jelöléseket, és értelmezéseket használni, melyet legtöbb ember egyszerűen megérthet. A programom funkciói elsősorban a közelítőmódszerekkel meghatározott függvények bemutatására szolgálnak, nem kimondottan konkrét helyek hőtérkép szerinti meghatározására.



Köszönetnyilvánítás

Az interpolációs témaválasztásommal kapcsolatban szeretnék köszönetet nyilvánítani Dr. Lócsi Leventének, a rengeteg érdekes téma felkínálásáért, ezenfelül a nagymértékű segítségért, amelyet hasznosítani tudtam a program megírása során. Szeretném továbbá megköszönni, hogy kitartott mellettem témavezetőként, és hogy felhívta figyelmemet a dolgozatban szereplő hibákra.

2. fejezet

Matematikai háttér

A program célja, hogy kapjunk egy olyan hőtérképet, amelyen láthatók mekkorák a hőmérsékleti értékek egyes helyek körül. A számításhoz interpolációs módszerek közül kell megválasztani az optimálisat, amely kellő pontossággal tudja közelíteni az általunk keresett mérési pontok közti terület hőmérsékleti értékét. A programhoz szükségem lesz eliminációs, azokat optimalizáló módszerekre, illetve interpolációs módszerre, amelyeket az alábbiakban részletezek.

2.1. Többváltozós interpoláció

A térképünkön lévő mérési pontokat a pixelek koordinátájával jelöljük. A kétdimenziós térképhez így szükségünk lesz egy többváltozós interpolációs módszerre. A lehelyezett alappontok nem szabályos formában helyezkednek el, így azokat szórt alappontoknak nevezzük, melyre jól működik a radiális bázisfüggvényekkel való közelítés. Ennek magas műveletigénye van, de látványosabb és pontosabb közelítéseket kaphatunk a többi módszernél.

2.2. Radiális bázisfüggvény

A többváltozós interpoláción belül a radiális bázisfüggvények eltoltjait fogjuk használni, ami segítségével megkaphatjuk a szükséges függvény közelítését. Minden egyes pixelt koordinátapontként kezelünk, és ezeket a közelítőfüggvény értéke szerint színezzük.

2.2.1. Definició (Radiális függvény)**.** Legyen $\varphi : [0, \infty) \to \mathbb{R}$ függvény. $\Phi : \mathbb{R}^n \to \mathbb{R}$ ($n \in \mathbb{N}$) függvényt radiális függvénynek nevezzük, ha $\Phi(\mathbf{x}) = \varphi(\|\mathbf{x}\|)$, ahol $\|.\| : \mathbb{R}^n \to [0, \infty)$ euklideszi norma.

Látható, hogy Φ értéke nem $\mathbf{x}$ függvényértékétől, hanem $\mathbf{x}$ pozíciójától függ. A továbbiakban az egyszerűség és szemléletesség kedvéért φ segítségével írom fel a szükséges definíciókat, illetve a programhoz szükséges kétváltozós számításnál $\mathbf{x} \in \mathbb{R}^2$-beli vektorokat használok.

A keresett függvényünk, ahol n jelöli az alappontok számát, a következő:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \, \varphi(\|\mathbf{x} - \mathbf{x}_i\|) \qquad (\mathbf{x} \in \mathbb{R}^m, n, m \in \mathbb{N}).$$

A keresett függvény kiszámításához továbbá szükség lesz $\alpha_1, \alpha_2 \dots \alpha_n$ értékekre, melyek az alappontok egymáshoz való viszonya szerint súlyozottak a hozzájuk tartozó φ függvényérték által. Ahhoz hogy $\alpha_1, \alpha_2 \dots \alpha_n$ értékeket megkapjuk, meg kell határoznunk az alappontokon felvett értékekre a lineáris kombinációjukat.

Tehát, legyenek $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in \mathbb{R}^m$-beli vektorok, melyek az alappontok helyét jelölik. Ekkor

$$\begin{aligned} f(\mathbf{x}_1) &= \sum_{i=1}^{n} \alpha_i \, \varphi(\|\mathbf{x}_1 - \mathbf{x}_i\|) \\ f(\mathbf{x}_2) &= \sum_{i=1}^{n} \alpha_i \, \varphi(\|\mathbf{x}_2 - \mathbf{x}_i\|) \\ &\dots \\ f(\mathbf{x}_n) &= \sum_{i=1}^{n} \alpha_i \, \varphi(\|\mathbf{x}_n - \mathbf{x}_i\|) \end{aligned}$$

lineáris egyenletrendszer megoldása az $\alpha \in \mathbb{R}^n$ vektor. A LER a következőképp írható fel.

$$\begin{bmatrix} \varphi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_1 - \mathbf{x}_2\|) & \cdots & \varphi(\|\mathbf{x}_1 - \mathbf{x}_n\|) \\ \varphi(\|\mathbf{x}_2 - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_2 - \mathbf{x}_2\|) & \cdots & \varphi(\|\mathbf{x}_2 - \mathbf{x}_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{x}_n - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_n - \mathbf{x}_2\|) & \cdots & \varphi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix}, \tag{2.1}$$

ahol

$$A = \begin{bmatrix} \varphi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_1 - \mathbf{x}_2\|) & \cdots & \varphi(\|\mathbf{x}_1 - \mathbf{x}_n\|) \\ \varphi(\|\mathbf{x}_2 - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_2 - \mathbf{x}_2\|) & \cdots & \varphi(\|\mathbf{x}_2 - \mathbf{x}_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{x}_n - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_n - \mathbf{x}_2\|) & \cdots & \varphi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{bmatrix} \tag{2.2}$$

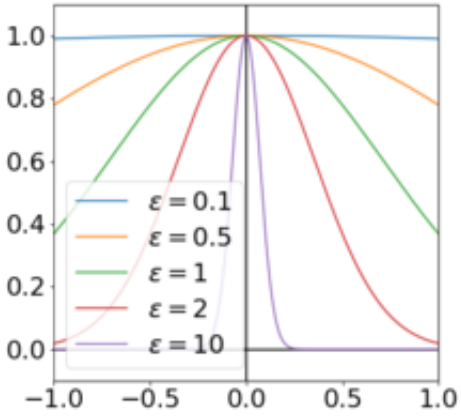
mátrixot interpolációs, vagy együttható mátrixnak nevezzük. [1]

2.3. Nevezetes bázisfüggvények

A radiális bázisfüggvényeknek számos fajtája létezik. Ezek közül ötöt használtam fel a program elkészítése során, melyeket részletesen szeretnék bemutatni.
A következő függvények vizsgálatához az alábbiakat kössük ki.

Rögzítsük $r = \|\mathbf{x} - \mathbf{x}_i\|$ paramétert, ahol $\|.\|$ szintén az euklideszi norma, ($i = 1, 2, \ldots, n$), ahol n az alappontok száma.

Továbbá bővítsük egy ε konstanssal, amely a közelítőfüggvény tompításához szükséges. Ez a konstans közvetlen r együtthatója lesz. Az ε-t az angolban *shape parameter*-nek, [2] magyarra fordítva *alakparaméter*-nek hívnak. Az ábrán jól látható hogy a Gauss függvény kis ε értékekre laposabb, míg nagy ε-ra meredeken, gyorsan közelít nullához.



2.1. ábra. Triviális egyváltozós Gauss bázisfüggvény különböző epszilon értékekre [3]

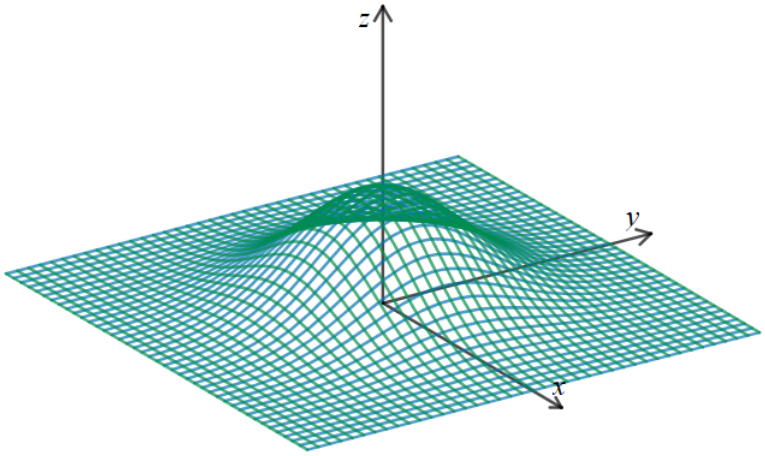
Az alábbiakban kifejtett függvényeknek illusztrációját online függvényábrázoló programmal készítettem. [4]

2.3.1. Gauss

A gauss függvény

$$\varphi(r) = e^{-(\varepsilon r)^2}$$

formájú, mely az alappontoktól távolodva a leggyorsabban közelít nullához.



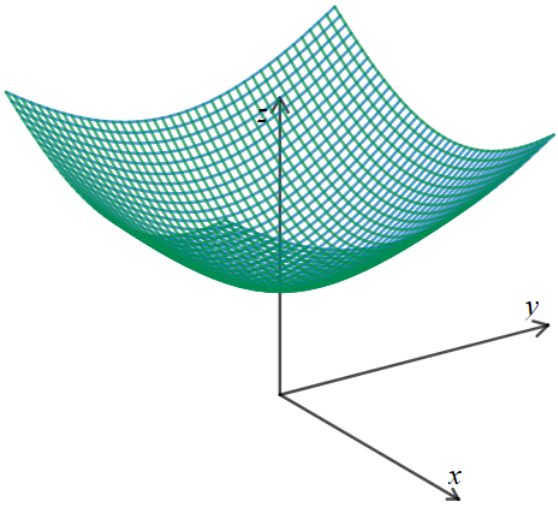
2.2. ábra. Gauss bázisfüggvény

2.3.2. Multiquadrics

A multiquadrics függvény

$$\varphi(r) = \sqrt{1 + (\varepsilon r)^2},$$

amely tart a végtelenbe az alappontoktól távolodva.

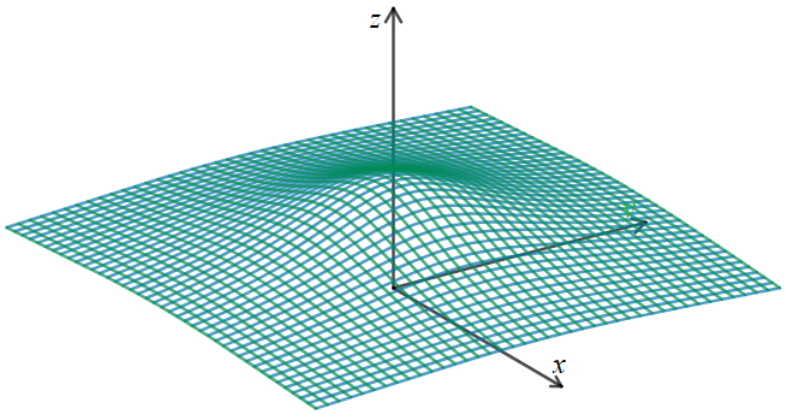


2.3. ábra. Multiquadrics bázisfüggvény

2.3.3. Inverz multiquadrics

Az inverz multiquadrics bázisfüggvény reciproka a multiquadrics függvénynek, így már nem végtelenbe hanem nullába tart.

$$\varphi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$$

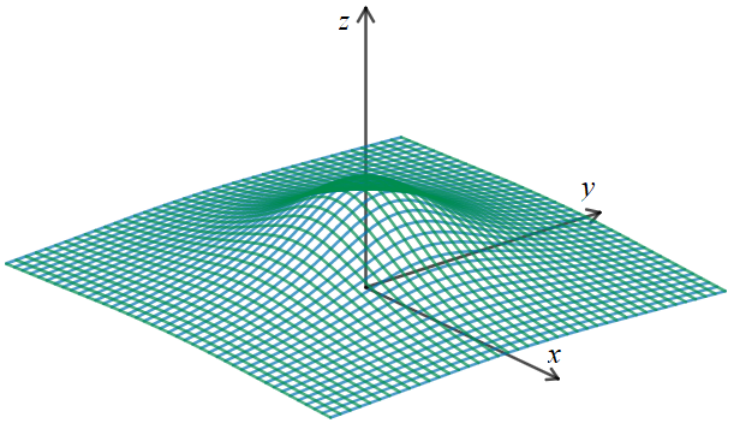


2.4. ábra. Inverz multiquadrics bázisfüggvény

2.3.4. Inverz quadratic

Az inverz quadratic bázisfüggvény reciproka a multiquadrics függvény négyzetének, így sokkal nagyobb mértékben közelít nullához mint az inverz multiquadrics bázisfüggvény.

$$\varphi(r) = \frac{1}{1 + (\varepsilon r)^2}$$



2.5. ábra. Inverz quadratic bázisfüggvény

2.3.5. Thin plate spline

A *thin plate spline* a *poliharmonikus spline*-ok egy speciális fajtája. Ahhoz hogy ezt lássuk írjuk fel a poliharmónikus spline függvényét.

$$\varphi(r) = r^k, \qquad k = 1, 3, 5, \ldots$$
$$\varphi(r) = r^k, \ln(r) \quad k = 2, 4, 6, \ldots$$

A poliharmonikus spline-oknál gondot okoz páros kitevős esetekben ($k = 2, 4, 6, \ldots$) a logaritmus, mert $r = 0$ esetben $\ln(0) = -\infty$. A probléma elkerülésére implementáláskor gyakran

$$\varphi(r) = r^{k-1} \ln(r^r)$$

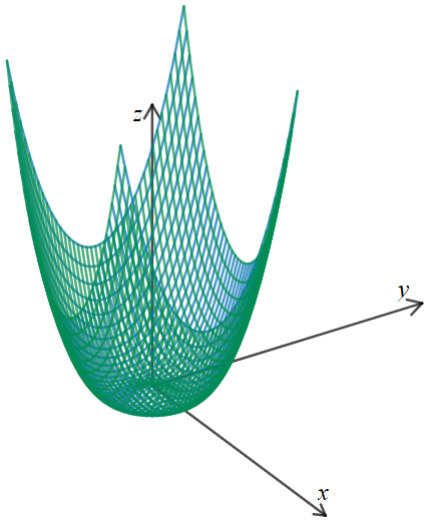
függvényt használják. A poliharmonikus spline nem tartalmazza ε alakparamétert, anélkül is megfelelő közelítést ad.

Thin plate spline a poliharmonikus spline $k = 2$-es esete.

$$\varphi(r) = r^2 \ln(r)$$

Az $r = 0$ esetén a korábban említett problémát úgy kerüljük el, hogy ezen a helyen rögzítjük, hogy a függvény értéke $\varphi(0) = 0$, egyéb esetekben pedig $\varphi(r) = r^2 \ln(r)$.

$$\varphi(r) = \begin{cases} r^2 \ln(r) & \text{ha } r \neq 0 \\ 0 & \text{ha } r = 0. \end{cases}$$



2.6. ábra. Thin plate spline bázisfüggvény

2.4. Lineáris egyenletrendszereket megoldó módszerek

A program számításához szükséges 2.2. fejezet (2.1) lineáris egyenletrendszerének megoldását hatékony numerikus módszerrel határozzuk meg. Tekintsük az A mátrixot a lineáris egyenletrendszer együttható mátrixának, $\mathbf{x} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix}^T$ és legyen $\mathbf{b}$ a lineáris egyenletrendszer eredményvektora. Ekkor a megoldandó probléma

$$A\mathbf{x} = \mathbf{b} \quad (A \in \mathbb{R}^{n \times n}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^n)$$

alakú, ha A invertálható mátrix, akkor

$$\mathbf{x} = A^{-1}\mathbf{b} \quad (A \in \mathbb{R}^{n \times n}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^n)$$

a lineáris egyenletrendszer megoldása.

2.4.1. Gauss-elimináció

A legismertebb módszer lineáris egyenletrendszer megoldására a Gauss-elimináció. Nagy műveletigényű ezért sok alappontra hosszú a futási ideje az algoritmusnak.

2.4.2. Gauss-elimináció részleges főelem kiválasztással

Amennyiben az A mátrix valamely i-re az $i-1$-ik eliminációs lépésre $a_{i,i} = 0$, részleges főelemkiválasztást alkalmazhatunk. Ebben az esetben az $i-1$-ik lépésnél szereplő mátrixnak megkeressük az adott oszlopban lévő legnagyobb elemek abszolút értékei közül a legnagyobbat ($|a_{i,i}|, |a_{i+1,i}|, \ldots, |a_{n,i}|$), melynek sorát felcseréljük az i-ik sorral.

Ezzel a módszerrel elkerülhetjük a kis számmal való osztást aminek köszönhetően csökkenthetjük a numerikus hibát. [5]

2.4.3. Cholesky-felbontás

A Cholesky-felbontással megoldható $A\mathbf{x} = \mathbf{b}$ egyenletrendszer ha az A mátrix szimmetrikus és pozitív definit. Az algoritmus gyorsasága miatt nem ellenőrizzük előre a definitséget, azt az algoritmus végrehajtásakor megkapjuk. A Cholesky-felbontás [6] lépései:

- A mátrixot felbontjuk alsó háromszögmátrixra, úgy hogy teljesüljön $A = LL^T$

egyenlőség.

- $LL^T\mathbf{x} = \mathbf{b}$, ahol $\mathbf{y} := L^T\mathbf{x}$
- Oldjuk meg $L\mathbf{y} = \mathbf{b}$ lineáris egyenletrendszert.
- A kapott $\mathbf{y}$-nal pedig az $L^T\mathbf{x} = \mathbf{y}$ egyenletrendszert.

$A = LL^T$ mátrixfelbontáshoz *Cholesky–Banachiewicz-algoritmust* használjuk, ahol

$$l_{i,j} = \frac{1}{l_{j,j}}(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k}l_{j,k}) \qquad (i > j)$$

$$l_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} l_{i,k}^2}$$

az L alsó háromszögmátrix elemei ezáltal számolható. Ha az algoritmusban szereplő gyök alatti tényező negatív, az algoritmus leáll. Az A mátrix ebben az esetben nem szimmetrikus pozitív definit.

3. fejezet

Felhasználói dokumentáció

3.1. Rendszerkövetelmények

A program futtatásához szükségünk van .NET keretrendszer 4.6.1 verziójára, melyet a Windows 7-től kezdődően minden Windows operációs rendszer támogat. Ezeknek az elvárt rendszerkövetelményét vettem figyelembe az alábbiaknál. [7] [8]

3.1.1. Minimum rendszerkövetelmények

Szoftverkövetelmények:

- **Operációs rendszer:** Windows 7/8/10 (x86)
- **.NET Framework:** 4.6.1

Hardverkövetelmények:

- **Processzor:** 1 GHz – *többmagos ajánlott*
- **RAM:** 1 GB – *2 GB ajánlott*
- **Tárhely:** 4,5 GB
- **Grafikus kártya:** 512MB

3.2. Telepítés

A program használata nem igényel telepítést, a futtatható állomány indításával munkára fogható. Ezen felül viszont rendelkezni kell .NET keretrendszer 4.6.1-es verziójával, melyet a felhasználó letölthet a Microsoft oldaláról.

3.3. Mentett fájlok szerkezete

Amennyiben a felhasználó saját inputfájlt szeretne generálni meglévő adataiból, fontos figyelembe vennie a fejezetben szereplő megkötéseket, hogy azoknak eleget téve készítse el a fájlokat. A fájlok saját kiterjesztéssel rendelkeznek. A program csak a kívánt kiterjesztésű fájlt képes megnyitni, így fontos ennek pontossága is.

Kétféle fájltípust különböztethetünk meg. A színskálát tároló fájl kiterjesztése *.mcf*, mely a *map color file* rövidítése, illetve az alappontokat tartalmazó fájl kiterjesztése *.mdf*, mely a *map data file* rövidítéséből ered.

3.3.1. Színskálafájl felépítése

A színskála tetszőleges számú színből állhat össze. Ezek egy sorban, szóközzel elválasztva szerepelnek a fájlban. Minden színhez négy adat tartozik, mely 0 és 255 közti egész szám lehet. Ezek a számok BGRA színtípusnak megfelelő adatok. Rendere az első szám a kék színhez tartozik, második a zöldhöz, harmadik a piroshoz, végül a színek átlátszóságáért felelős alfa adattag.

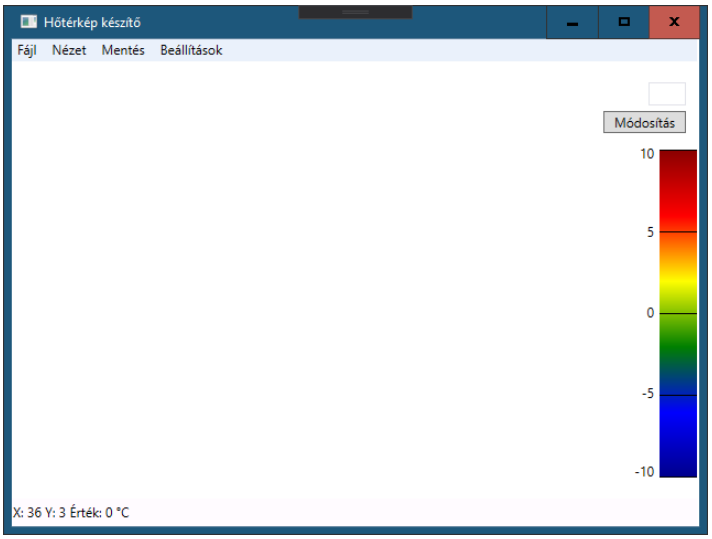
3.3.2. Alappontfájl felépítése

Az alappontfájl felépítése egyes mérések importálásához szükséges. A fájl szintén egy sorban tárolja az egyenként három adattagot tartalmazó alappontokat. Az alappont első tagja az x tengelyen, második az y tengelyen felvett pixelérték, a harmadik pedig a függvény, vagy hőmérsékleti érték. Mindhárom egész szám.

3.4. Felületi leírás, használat

A program indításkor a főablakot jeleníti meg, amely még nem tartalmaz alappontokat. Felül található egy menüsor, ahol ki lehet választani a számunkra megfelelő beállításokat,

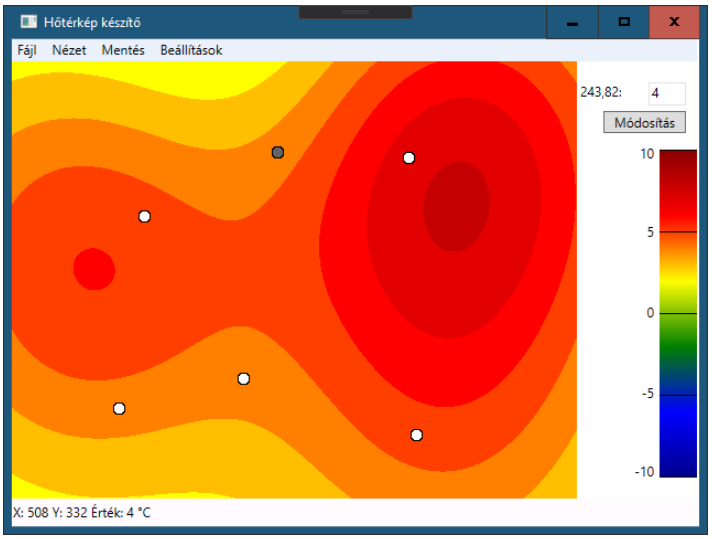
mentés menüpont alatt betölthetünk korábbi mentéseket, nézet alatt módosíthatjuk a színskálát, és a későbbiekben megtekinthetjük a függvényünk háromdimenziós ábrázolását.



3.1. ábra. Üres főablak kinézete

3.4.1. Főablak kezelése

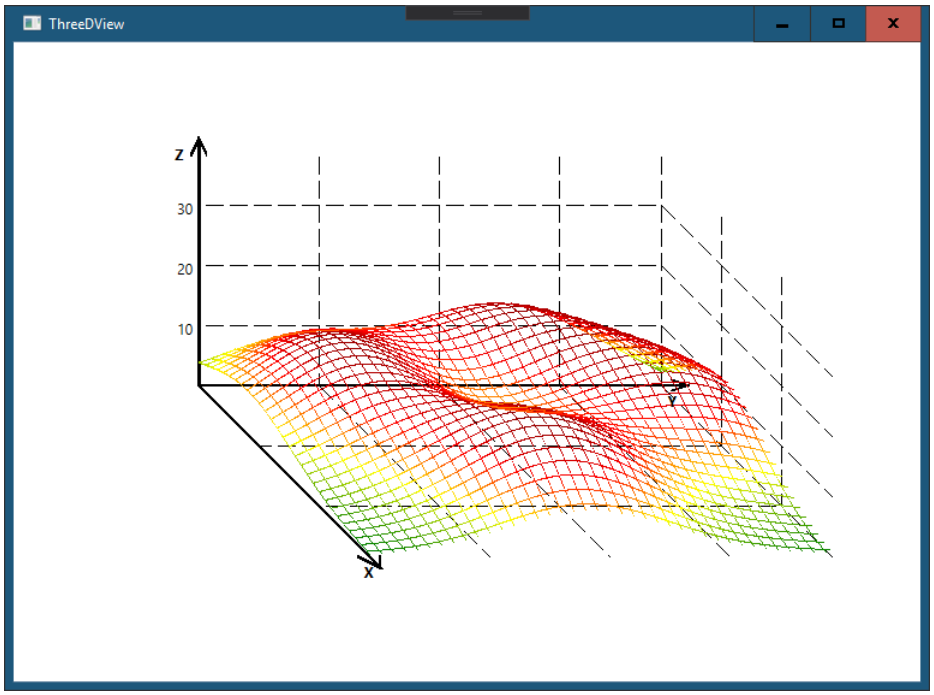
Ahhoz hogy ábrázoljon a főablak lehelyezhetünk pontokat, melyet a vászonra a bal egérgomb dupla kattintásával tudunk megtenni. Ilyenkor a lehelyezett pont értéke 0 °C. Egy pontot kijelölve módosíthatjuk értékét a jobb oldalon megjelenő szám átírásával, vagy az egérgörgő mozgatásával. A kijelölt pontot, – amely szürke színnel megkülönböztetett – lehetséges törölni a **Delete** gombbal.



3.2. ábra. Főablak lehelyezett alappontokkal

3.4.2. Háromdimenziós függvényábrázolás

Ezen az ablakon látható a koordináta rendszer, és azon elhelyezve a közelítőfüggvényünk, aminek tulajdonságait könnyebben vizsgálhatjuk.



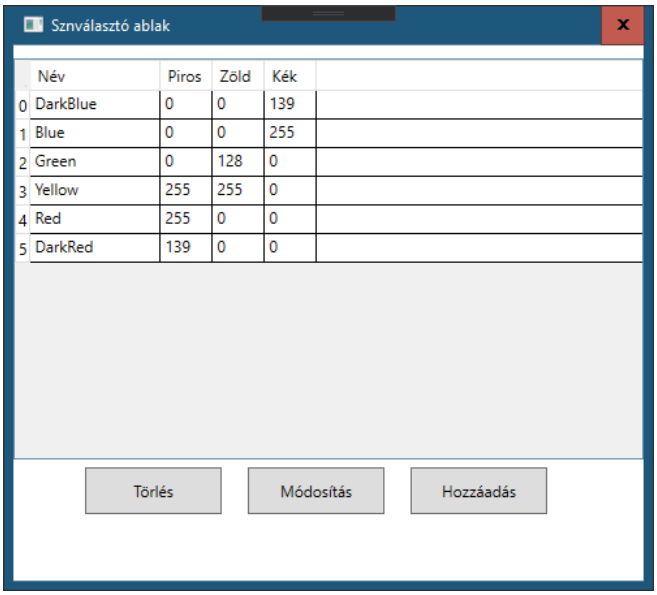
3.3. ábra. Gauss-függvényekkel való közelítés

3.4.3. Színskála módosítás

A színskála módosítóablakát a „Nézet” menüpont alatt érhetjük el, vagy a **Ctrl+1** billentyűkombinációval.

A színskálához hozzá tudunk adni új színt, vagy már létezőt módosíthatunk kívánalmunknak megfelelő gombra kattintva, amely beépített színválasztó ablakot jelenít meg, amelyen tetszés szerint választhatunk színt. Ha már létezik az adott szín abban az esetben a program nem adja hozzá a színskálához, ezt információs ablakban jelzi. 3.11. ábra.

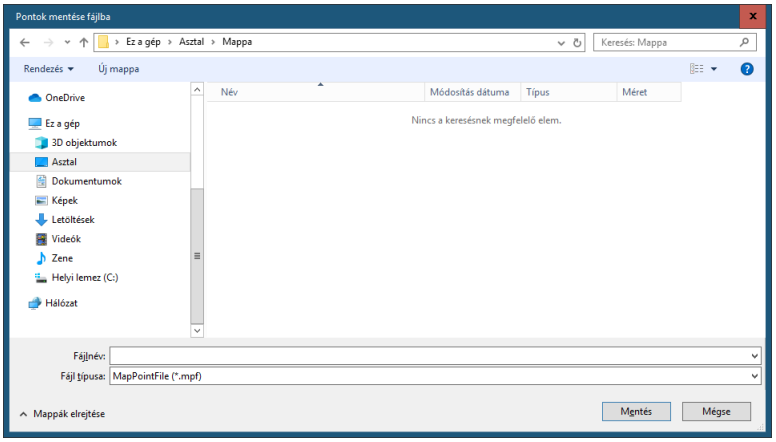
Amennyiben törölni szeretnénk, jelöljük ki a törlendő színt, majd a „Törlés” gombbal törölhetjük azt. Fontos megemlíteni, hogy két színnél kevesebb nem lehet a táblázatban, így két színnél már nem enged törölni a program.



3.4. ábra. Színskála módosítóablaka

3.4.4. Kép, pont, színskála mentése

A pontokat, színskálát, és a már ábrázolt hőtérképet azonosan a „Mentés” legördülő menüsáv alatti menüpontokra kattintva lehet menteni, vagy betölteni. A mentésnél mindháromnál ugyanaz a mentőfelület található, csupán a mentendő fájl kiterjesztésében térnek el. Ez a mentőablak a C# beépített osztályába tartozó dialógusablak. Ugyanez jelenik meg az új térkép kezdésekor elmentendő pontok esetén. A mentendő fájl nevét az annak megfelelő helyre írva menthetünk a kiválasztott mappába, később a fájlt ugyanitt találjuk meg betöltés esetén. A színek betöltésénél, illetve a pontok betöltését kezelő betöltő dialógusablak hasonlít az előzőhöz, csak olyan fájlokat jelenít meg aminek a kiterjesztése megegyezik az általunk elvárt kiterjesztéssel.



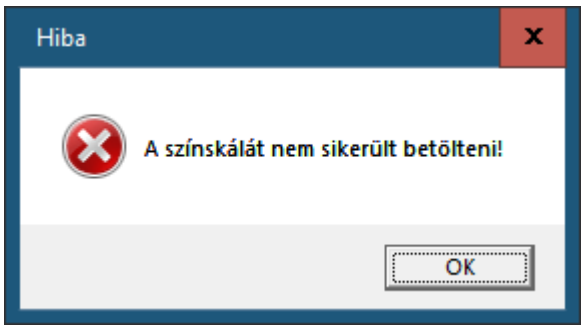
3.5. ábra. A mentésekhez tartozó felület

3.5. Hibaforrások, hibaüzenetek

A program futása során felmerülő felhasználói hibalehetőségek, a fejezetben bemutatott hibák esetén kezelve lettek. Célja, hogy a felhasználó visszajelzést kapjon, hogy adott problémára reagálva, a programot helyesen használja. A hibaüzenetek felugró ablakon jelennek meg.

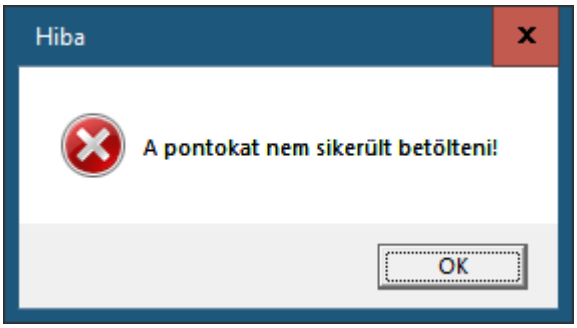
3.5.1. Betöltéskor, mentéskor adódó hiba

Amikor a színek betöltésénél, a forrásfájlban nem várt érték található, akkor a következő hibaüzenetet kapjuk.



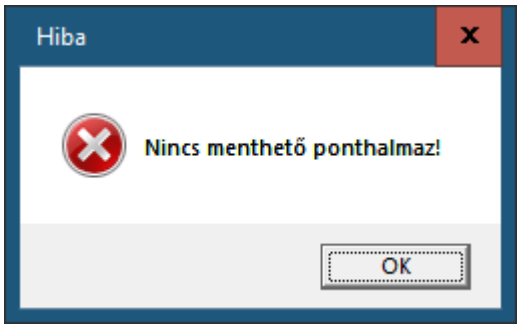
3.6. ábra. Színbetöltés hibaüzenete

Amennyiben az alappontok betöltésénél merül fel hiba, a program a következő üzenetet írja.



3.7. ábra. Alappontbetöltés hibaüzenete

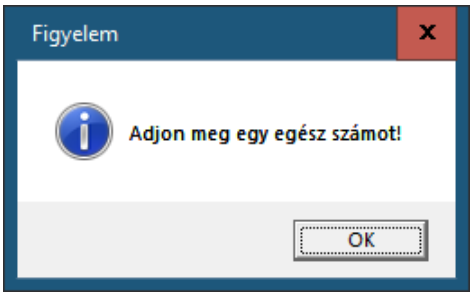
Abban az esetben ha üres térkép nem létező alappontjait próbáljuk elmenteni, a program szintén hibát jelez.



3.8. ábra. Alappontmentés hibaüzenete

3.5.2. Rossz adat megadása

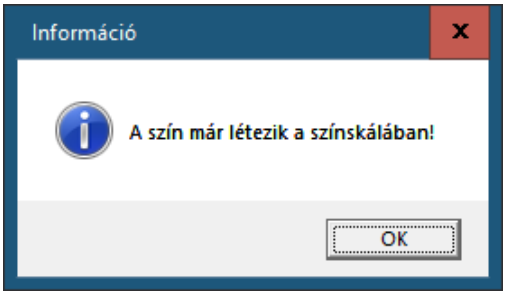
Ha a kiválasztott alappontnak szeretnénk más értéket adni, de annak típusa nem egész, abban az esetben a program figyelmeztet egy információs ablakban, hogy az adatot megfelelően adjuk meg.



3.9. ábra. Helytelen adat információs ablaka

3.5.3. Létező szín kiválasztása

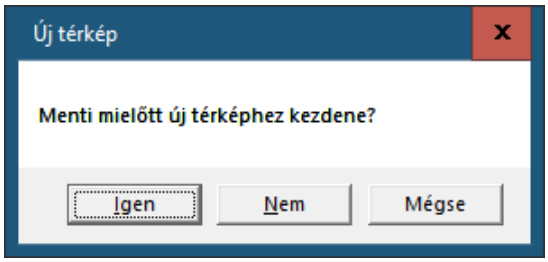
Amikor a színskálán szeretnénk módosítani, vagy ahhoz színt hozzáadni, nem lehetséges már a skálában létező színt a táblázathoz adni. Ilyenkor a program információs ablakban közli a problémát.



3.10. ábra. Létező szín információs ablaka

3.5.4. Új térkép kezdése

Amennyiben rákattintunk az „Új térkép” menüpontra kattintva a program felajánlja, hogy eddigi munkánkat menthessük, az ne vesszen el. Ekkor három lehetőség közül lehet választani, amennyiben a nemre kattintunk, üres vászont nyit nekünk a program, ha igenre kattintunk, abban az esetben felugrik a mentés felület ahol a pontokat tudjuk menteni. A mégse gombbal pedig visszatérhetünk jelenlegi munkánkhoz.



3.11. ábra. Szándékellenőrző felület

4. fejezet

Fejlesztői dokumentáció

4.1. Megvalósítandó feladat

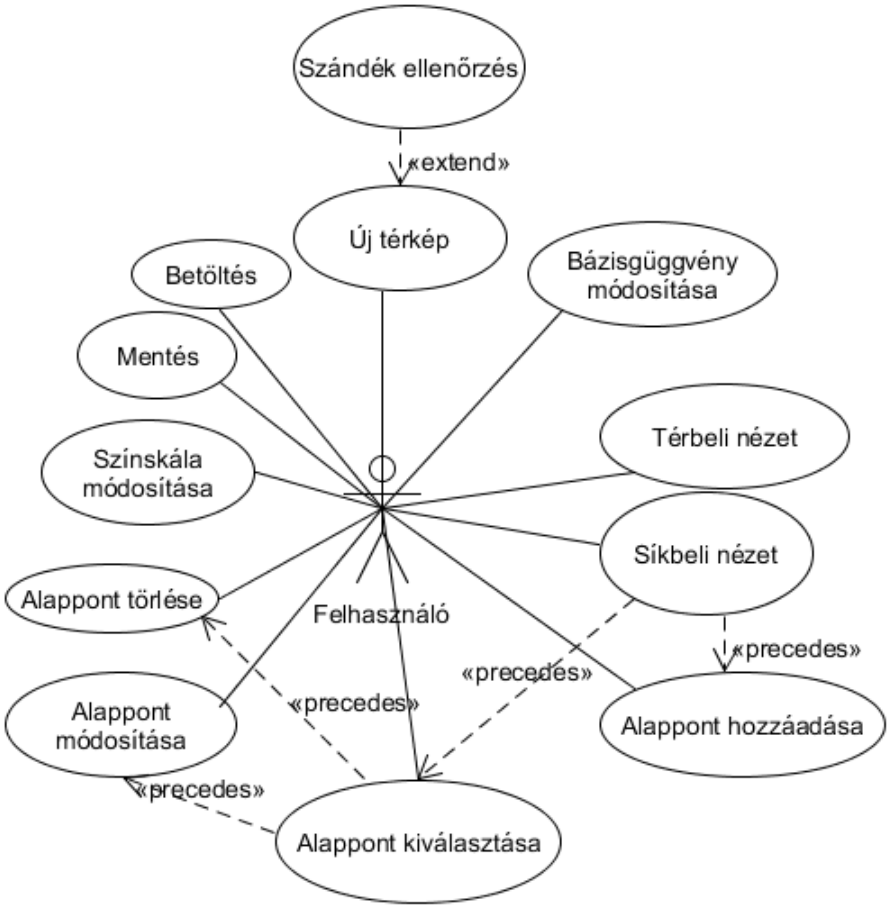
A program előre megadott alappontok beszúrásával, az alappontokon mért adatokkal számoljon. Radiális bázisfüggvények eltoltjai megfelelő lineáris kombinációjának meghatározása által színezze be a térkép teljes felületét, ezzel közelítést adva az alappontok közti területek hőmérsékletére. Ehhez a részhez lesz szükségünk a matematikai eszközökre, hogy gyorsan és pontosan határozzuk meg a térképet. Nyújtson lehetőséget a színskála tetszés szerinti megválasztására. Ehhez, egyszerű lineáris színátmenetet használok a két alapszín közti értékek számításához. Továbbá háromdimenziós nézet funkció megvalósítása is cél, ahol térbeli ábrázolását láthatjuk az illesztett függvénynek. A lehelyezett alappontok pozícióját lehessen menteni, az adatok későbbi használatának megkönnyítése végett, emellett lehetőséget adunk a színskála mentésére is.

4.1.1. Használati esetek

A program követelményleírása szerinti használati esetdiagram, ahol a lényeges funkciók kerülnek ábrázolásra, melyeket a felhasználó elér és használhat futás közben.

4.1.2. Felhasználói történet táblázat

A táblázatban a megvalósítandó program funkcióinak részletes bemutatása szerepel felhasználói szemszögből. A követelmény szerinti lehetséges esetekre kitérve a program válaszát láthatjuk az adott szituációkban. A felhasználói eset alatt találhatók a leírásban szereplő funkciók nevei. A leírás alatt pedig egy GIVEN-WHEN-THEN hármas mutatja



4.1. ábra. Felhasználói esetdiagram

minden pontnál, hogy az adott programállapotban, a program, felhasználói beavatkozásra, milyen műveletet hajt végre. Jelen esetben az angol szavakat ADOTT-AMIKOR-AKKOR szavakra cserélem az értelmezés megkönnyítése miatt.

#	Felhasználói eset	Leírás	
1.	**Alkalmazás indítása**	ADOTT	Az alkalmazás Windows 7 vagy újabb operációs rendszeren telepítve van,
		AMIKOR	alkalmazás indítása,
		AKKOR	megjelenik az üres vászon.
2.	**Kilépés**	ADOTT	Az alkalmazás fut,
		AMIKOR	bezáró ikonra kattintunk,
		AKKOR	az alkalmazás leáll.

3.	**Új térkép**	ADOTT	Az alkalmazás fut,
		AMIKOR	„Fájl » Új térkép” menüpontra kattintunk,
		AKKOR	az alkalmazás szándékunk szerint menti a jelenlegi térképet, majd új vásznat ad.
4.	**Alappont lehelyezése**	ADOTT	Az alkalmazás fut,
		AMIKOR	a vászonra duplán kattintunk,
		AKKOR	az alkalmazás lehelyez egy alappontot és kiszámítja a közelítőfüggvényt.
5.	**Alappont értékének módosítása**	ADOTT	Az alkalmazás fut,
		AMIKOR	rákattintunk egy alappontra,
		AKKOR	az alappont kijelölődik melyet görgővel, vagy érték beírásával módosíthatunk.
6.	**Alappont törlése**	ADOTT	Egy alappont kijelölve,
		AMIKOR	Delete gombot megnyomjuk,
		AKKOR	az alappont törlődik, az alkalmazás újraszámítja a térképet.
7.	**Adott hely értéke**	ADOTT	Az alkalmazás fut,
		AMIKOR	az egér kurzorral a vászonra navigálunk,
		AKKOR	a program az alsó állapotsávban jelzi az adott pontot és azon felvett függvényértéket.
8.	**Térképszínezés módosítása**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Nézet” legördülősáv „Színek megválasztása” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a színskála módosító ablakot.

9.	**Szín módosítása**	ADOTT	Színskála módosító ablak nyitva, kijelölve egy szín a táblázatban,
		AMIKOR	„Módosítás” gombra kattintunk
		AKKOR	az alkalmazás megnyitja a színválasztó ablakot, ahol kiválaszthatjuk milyen színre szeretnénk módosítani.
10.	**Szín törlése**	ADOTT	Színskála módosító ablak nyitva, kijelölve egy szín a táblázatban,
		AMIKOR	„Törlés” gombra kattintunk
		AKKOR	az alkalmazás kitörli az a kiválasztott színt.
11.	**Szín hozzáadása**	ADOTT	Színskála módosító ablak nyitva
		AMIKOR	„Hozzáadás” gombra kattintunk
		AKKOR	az alkalmazás megnyitja a színválasztó ablakot, ahol kiválaszthatjuk milyen színre szeretnénk hozzáadni a skálához
12.	**Térbeli nézetre váltás**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Nézet” legördülősáv „3D nézet” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a 3D nézetet a jelenlegi hőtérkép kirajzolásával.
13.	**Térképvászon mentése**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Mentés” legördülősáv „Kép mentése” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a mentés ablakot ahol kiválaszthatjuk hova mentsük a képet.

14.	**Alappontok mentése**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Mentés” legördülősáv „Pontok mentése” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a mentés ablakot ahol kiválaszthatjuk hova mentsük a pontokat.
15.	**Alappontok betöltése**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Mentés” legördülősáv „Pontok betöltése” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a tallózó fájlt, ahol megkereshetjük korábbi mentésünk, és azt betölthetjük.
16.	**Színskála mentése**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Mentés” legördülősáv „Színskála mentése” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a mentés ablakot ahol kiválaszthatjuk hova mentsük a színeket tároló fájlt.
17.	**Színskála betöltése**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Mentés” legördülősáv „Színskála betöltése” menüpontjára kattintunk,
		AKKOR	az alkalmazás megnyitja a tallózó fájlt, ahol megkereshetjük korábbi színskálánkat, és azt betölthetjük.
18.	**Mátrixfelbontás algoritmusa**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Beállítások” legördülősáv „Mátrix felbontás” menüpontjánál kiválasztjuk a tetszőleges felbontást,
		AKKOR	az alkalmazás újraszámítja a térképet a kiválasztott algoritmussal.

19.	**Bázisfüggvény típusa**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Beállítások” legördülősáv „Bázisfüggvény típusa” menüpontjánál kiválasztjuk a tetszőleges függvénytípust,
		AKKOR	az alkalmazás újraszámítja a térképet a kiválasztott bázisfüggvénnyel.
20.	**Függvényfolytonosság**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Beállítások” legördülősáv „Függvényfolytonosság” menüpontjánál kiválasztjuk, hogy folytonos vagy kerekített legyen a függvény,
		AKKOR	az alkalmazás újraszámítja a térképet a kívánt módon.
21.	**Epszilon értékének kiválasztása**	ADOTT	Az alkalmazás fut,
		AMIKOR	a „Beállítások” legördülősáv „Epszilon értéke” menüpontjánál megválaszthatjuk a megadott értéket,
		AKKOR	az alkalmazás újraszámítja a térképet a megválasztott alakparaméterrel.

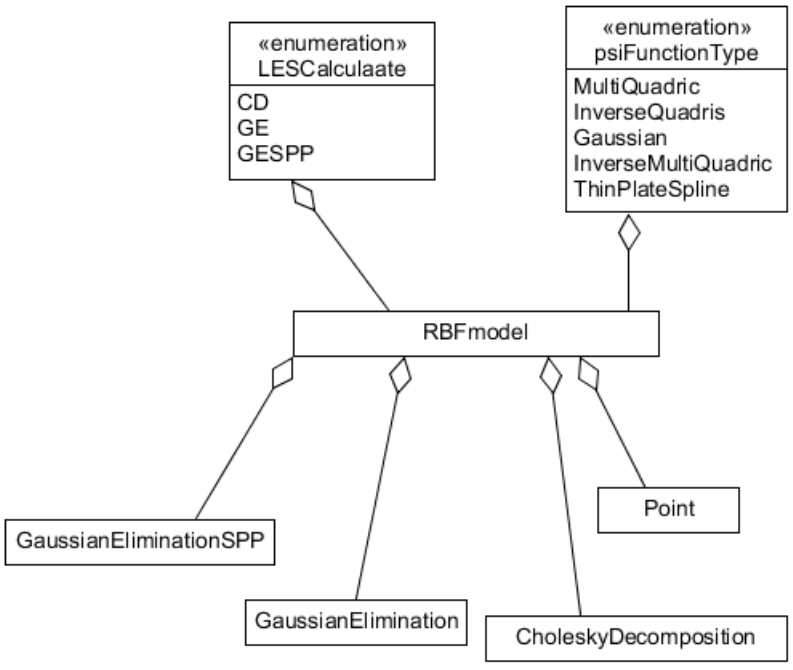
4.2. Megvalósítás

4.2.1. Fejlesztői környezet, architektúrák

A programot C# nyelven késztettem WPF GUI-val. Ennek követelménye Windows operációs rendszer, így a programot csak ezzel a rendszerrel rendelkező felhasználók tudják használni. Előnye a Windows Forms alkalmazásokkal szemben hogy rugalmasabban, és átláthatóbban kezelhető az XAML fájlnak köszönhetően.

Az MVVM architektúra megkülönbözteti, és elválasztja a *modell*, *nézet* és *nézetmodell* rétegeit, emellett külön rétegben implementálható a perzisztencia ami mentéseknek, és betöltéseknek ad helyet. Ezeknek köszönhetően, lesznek szemléletesebbek a kódok, és könnyebben átláthatók.

4.2.2. Modell réteg osztályai



4.2. ábra. Modell réteg osztálydiagramja

Point osztály

A **Point** osztály adattagjai közül x az x, míg y az y tengelyen felvett értéket, h pedig az (x, y) koordinátában felvett hőmérsékleti értéket jelöli. Az osztály konstruktora **Point**.

A konstruktor három `int` típusú paramétere adódik át az adattagoknak, ezekben tárolja a pont értékeket.

CholeskyDecomposition osztály

Ez az osztály Cholesky-felbontásban számolja az egyenletrendszer megoldását. Az 2.4.3. fejezetben leírt módszer algoritmusa került implementálásra, amely a konstruktorban végre is hajtódik. A konstruktor két bemenő paramétert vár. Ezek típusa `double[,]` kétdimenziós tömb, amely az együttható mátrix, illetve egy `double[]` tömb, a lineáris egyenletrendszer eredményvektora. Ehhez szükséges egy mátrixtranszponáló függvény, amely az alsó háromszögmátrix transzponálására szolgál, ez a **Transpose** nevet viseli. Bemenő paramétere egy `double[,]` típusú két dimenziós tömb, egyben a függvény típusa is. Tartalmaz egy függvényt, mely a példányosítás után meghívható **GetX** névvel. A függvény `double[]` típusú tömböt ad vissza értékül, amely az egyenletrendszer megoldása, melyet egy azonos típusú privát adattagban tárolunk, ez a konstruktor meghívásakor számolódik.

GaussianElimination osztály

A Gauss-elimináció algoritmusának megvalósítását ezen osztályon belül végezzük, ahol a konstruktor paraméterei az előzővel azonosak, az osztály szintén tartalmaz egy privát *x* adattagot, és egy **GetX** függvényt, melynek visszatérési értéke az előzőével azonos. A konstruktorban számoló algoritmust [9] forrásban lévő algoritmus szerint valósítottam meg. Az osztály abban az esetben, ha 0 értékkel találkozik a hányados nevezőjében, a *GaussianEliminationSPP* osztályt hívja segítségül, amelynek ez már nem okoz problémát például a *thin plate spline-ok* együtthatómátrixánál.

GaussianEliminationSPP osztály

A **GaussEliminationSPP** osztály az előzővel azonos függvényt és adattagot tartalmaz, az algoritmus a [9] oldalon lévő *Scaled Partial Pivoting* algoritmus szerint valósítottam meg, amely a részleges főelem kiválasztást jelenti.

RBFmodel osztály

Az **RBFmodel** osztály konstruktor `IRBFPersistenceDataAccess` típusú bemenő attribútuma a perzisztencia rétegben lévő interfész amit a kapcsolunk az osztályhoz, a privát *_dataAccess* szintén azonos típussal rendelkező adattagban tároljuk.

Az osztályhoz tartozó privát adattagok közé tartozik a `LESSolve`, amely a `LESCalculate` felsorolóosztály egy tagját tárolja. Ennek kezdőértéke a jelenlegi osztály példányosításakor *GE*, amely a gauss-elimináció rövidítése. Ezt az értéket **ChangeFT** nyilvános metódus segítségével tudjuk változtatni, melynek paramétere szintén a felsoroló osztálybeli. Másik a *BasisFunctionType*, ami a bázisfüggvény típusára vonatkozik, melyeket a `psiFunctionType` felsoroló osztály értékei közül választunk, hozzá tartozó eljárás, mely az értékét változtatja a **ChangeRBFType**, mely ezen felsoroló osztálybeli paraméterrel rendelkezik. Alapértelmezett értéke *Gaußian*.

Az **AddPoint** az osztályhoz tartozó metódus, amely három paraméterrel rendelkezik. Mindhárom `double` típusú, a **Point** osztályhoz szükséges változóértékek. A metódus hozzáadja a pontot a *points* listához, ha az még nem létezett benne, vagy nincs egy másik ponttól való 10 pixelnyi távolságon belül. Amennyiben a *points* lista nincs példányosítva mivel a pontok száma nulla, abban az esetben példányosítja. Ennek a metódusnak a párja a **RemovePoint** függvény, amelynek visszatérési értéke logikai `bool` típus, ez a későbbiekben lesz fontos. Amennyimen sikerül eltávolítani híváskor a pontot, *igaz* értékkel tér vissza. Ellenkező esetben pedig *hamissal*.

A **NewMap** metódus paraméter nélküli eljárás, amely a *localMinimum* és *localMaximum* privát adattagokban lévő értéket visszaállítja a kezdőértékre, a *points* tömböt és ha aktuálisan volt kijelölt pont, mindkettőt felszabadítja.

Pontok módosítására három eljárást alkalmazhatunk. Ezek az eljárások visszatérési érték nélküliek, illetve a **PointIncrease** és **PointDecrease** metódusok, melyek közül az első a kiválasztott pont értékét növeli, a második csökkenti, nem rendelkeznek bemenő paraméterrel az adott pont melynek értékét módosítani kell a *_choosedPoint* adattagban tárolt pont. A harmadik metódus **PointValue** amely egy egész bemenő értékkel rendelkezik, amely a kívánt hőmérsékleti érték. Ezt adjuk át szintén a *_choosedPoint-tal* megegyező koordinátákkal rendelkező pontnak, amennyiben van kiválasztott pont.

A függvényértékeket számító metódusokhoz tartozik a **CalculateMatrix** eljárás, amely nem rendelkezik bemenő paraméterrel. Megalkotja az interpolációs mátrixot, melyben minden pontpárra meghívja a **Phi** függvényt, amely 3 bemenő paraméterrel rendelkezik. A két első Point, míg a harmadik `psiFunctionType` típusú attribútum. A függvényben itt vannak implementálva a bázisfüggvények. Ehhez használjuk az osztály *epsilon* adattagját, amely az alakparaméter epszilonnak felel meg, és az **EuclideanNorm** double visszatérési értékű függvényt, amelynek egy pont a bemenő paramétere és nevének megfelelően az euklideszi normát számítja. A **CalculateMatrix** eljárás végül meghívja és *w* adattagnak értékül adja a **LinearEquationSystemCalculate** függvényt, amely a három egyenletrendszert megoldó algoritmus közül meghívja az aktuálisan kiválasztottat.

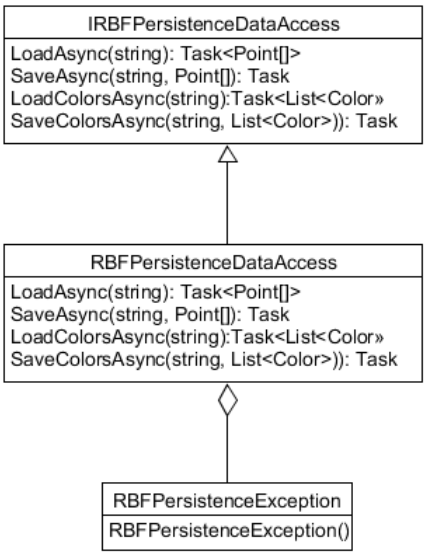
A **FunctionValue** metódus segítségével tudjuk meg egy szükséges pontra az interpo-

lációs függvény értékét, erre az alábbi eljárásokban lesz szükség. A magasabb rétegekben lévő vásznak számításához szükséges függvényekhez szükségünk lesz a függvényértékek maximumára, illetve minimumára. Ehhez használjuk a **FindlocalMaximumand-Minimum** függvényt, melynek visszatérési értéke double[] tömb, melynek első értéke a minimumérték, második a maximumérték. A függvényfolytonosságtól függően, melyet a *cab* adattagban tárolunk, számítjuk a kerekített, vagy pontos függvényértéket az adott pixelen. A **GetColors** függvény *System.Windows.Media.Imaging* névtérben lévő `WriteableBitmap` típusú függvény. Két bemenő egész értékű paraméter szélesség és magasság, amely egy fentről érkező vászon méreteinek felelnek meg. A későbbiekben is használatos *pixels1d* egydimenziós `byte` típusú elemeket tartalmazó tömbbe mentjük el a színeket. Az *fxy* lokális változóba számoljuk hogy az *i*-edik sorban, és egyben ciklusiterációban *i* milyen messze található a vászon tetejétől. Ez alapján színezi a vásznat a függvény. Mivel a **GetMap** metódus által számolt kép színskálájának ábrázolásához használjuk ezt a függvényt, így a már abban egyszer lefutó **FindlocalMaximumand-Minimum** függvény értékeit használjuk, a korábban már említett *localMaximum*, illetve *localMinumum*-ban eltárolva. Így nem kell kétszer végigmenni a térképen hogy ezt az értéket tudjuk. Jelenlegi kódban ez gondot nem okot, esetleges továbbfejlesztésnél erre viszont figyelni kell. A **GetMap** függvény szintén `WriteableBitmap` típusú, és előzővel azonos bemenő paraméterekkel rendelkezik. Amennyiben nincsenek lehelyezett pontjaink *null* értékkel tér vissza. Ellenkező esetben viszont kiszámolja a lineáris egyenletrendszert, a lokális minimum és maximum értéket, Majd az előző függvénnyel közel azonos módon konstruálja meg a bitmapot. Eltérés az *fxy* változó számolásában van. Az adott ponton lévő függvényértéket a folytonos függvény esetén **FunctionValue**-t jelenlegi ciklusiterációk *i*, illetve *j* értékével hívjuk meg. Nem folytonos esetben kerekítjük az értéket. A **WritePoints** függvény berajzolja a pontokat a tömbbe. A **SelectedPoint**, pedig a kiválasztott pontot színezi szürkére, és a *_choosedPoint* értékét ráállítja a pontra. A **GetThreeDMap** függvény két feladatot lát el. Első sorban kirajzolja a koordináta rendszert melybe a háromdimenziós ábrázolását tesszük a közelítőfüggvénynek. Tengelyenként ezt külön teszi meg az algoritmus. Ezután kiszámítjuk a függvényt, és elhelyezzük a vászonhoz szükséges tömbben.

A mentésekkel foglalkozó eljárásokat három fele választhatjuk. Az első kettő a színek, illetve mentésével és betöltésével foglalkozik. A **LoadColors** eljárás, mely egy fájl elérési útvonalát tartalmazza egyetlen paramétereként, ha nincs csatlakoztatott adatkapcsolat, abban az esetben visszatér, és nem kéri be a színeket, ha viszont megfelelően kapcsoltuk a perzisztenciát, akkor a colorize privát színeket tartalmazó listába betölteti a fájlt. A **LoadPoints** törli a léterő pontokat, majd új listát készít a betöltött pontokból, ezután meghívja a korábban leírt **CalculateMatrix** eljárást. A **SaveColors**, illetve **SavePoints** eljárások a colorize és points listákat adják át a perzisztencia megfelelő metódusainak.

Mindkét eljárás bemenőparaméterként a fájl elérési útvonalát névvel együtt kapják meg. A **saveMap** eljárás másként működik mint az előző két esetben lévő mentés. Két bemenő paramétere közül az első a `Canvas` típusú vászon, melyet a későbbiekben láthatunk, hogy a kirajzoltatott térkép. Második attribútum az előzőkkel azonos karakterlánc amely szintén az útvonalat tartalmazza.

4.2.3. Perzisztencia réteg osztályai



4.3. ábra. Perzisztencia réteg osztálydiagramja

IRBFPersistenceDataAccess interfész

A réteghez tartozó interfész, amely összeköttetést tud létesíteni más osztályokkal, jelen esetben a modell rétegben lévő **RBFmodel** osztállyal, az Interfészben deklarált eljárások segítségével. Jelen esetben 4 aszinkron eljárás a LoadAsync, SaveAsync, LoadColorsAsync, és a SaveColorsAsync, melyeket a következő alfejezetben fejtek ki, ahol a definíciójuk is található.

RBFPersistenceDataAccess osztály

A **LoadAsync** eljárásban *StreamReader* osztály segítségével olvassuk be aszinkron a sorokat. Ez azt jelenti hogy minden sor külön szálként olvasódik be, ezzel gyorsítjuk a beolvasást. Ezután eltároljuk a számokat, amiket a sorok felszeletelésével kapunk. Beletesszük a kapott számokból megkonstruált pontot a tömbbe, amelyet a végén visszaadunk.

A **SaveAsync** eljárás a pontok mentését végzi. *StreamWriter* osztály segítségével nyitjuk meg az írandó fájlt. A megnyitás után a pontok adattagjait szóközökkel elválasztva írjuk fájlba.

A **LoadColorsAsync** eljárásban a színskálát töltjük be fájlból. Amint a fájlban szereplő számok száma nem éri el a 8-at, akkor hibát dobunk, mivel a színskálát nem tudjuk felállítani két szín nélkül. Szintén a *StreamReader* beépített osztály segítségével végezzük a beolvasást.

A **SaveColorsAsync** eljárásban a színskálának a mentését végezzük. A kimenő listának színeinek adattagjait egyesével vesszük, karakterlánccá konvertáljuk. Az adattagokat *BGRA* sorrendben írjuk ki a fájlba szóközzel elszeparálva.

A most leírt eljárások mindegyikében szerepel a „try-catch" szerkezet, kivételkezelés céljából. Amint nem sikerül a betöltés valami miatt, vagy az említett hosszprobléma miatt a z eljárás kivétellel tér vissza, amit a modell rétegben le tudunk kezelni.

RBFPersistenceException osztály

Exception osztályból leszármaztatott osztály, melyet a hiba elkapásánál használunk, és az előző alfejezetben definiált eljárások alatt keletkező hiba esetén dobjuk.

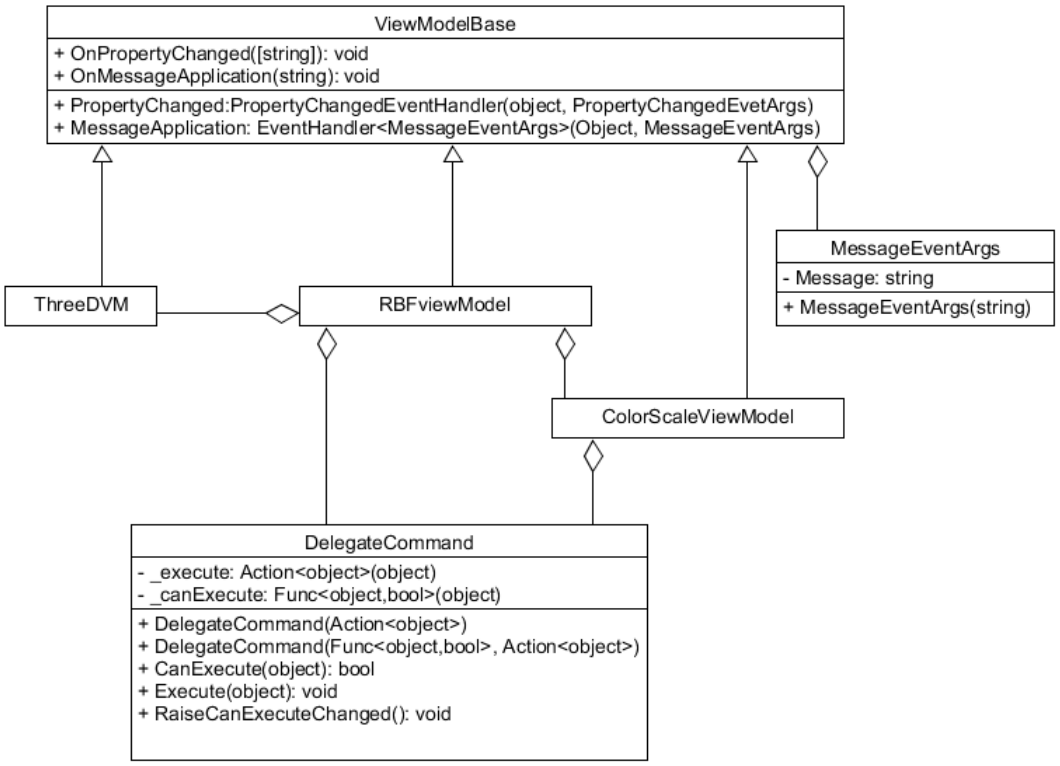
4.2.4. Nézet-modell réteg osztályai

MessageEventArgs osztály

Az **EventHandler** metódushoz szükséges, az üzenetet tartalmazza. Privát adattagja karakterlánc típusú.

ViewModelBase osztály

Az osztály számunkra nagyon fontos. Ebben található az **OnPropertyChanged** metódus, amely jelzi hogy valamelyik számunkra szükséges adattag változott. Illetve itt található az **EventHandler** metódus is ami az eseménykezelés fontos eljárása. Mindhárom saját nézet-modell osztályt ebből származtatjuk le.



4.4. ábra. Nézet-Modell osztálydiagramja

DelegateCommand osztály

A nézetet köti a nézet-modell rétegben lévő osztályainkhoz. Ezeken keresztül küldjük át az egyes gombok, vagy menüpontok által kiváltott eseményeket.

ThreeDVM osztály

A **ThreeDVM** osztály példányosításkor paraméterként kapja a hozzá tartozó nézet rétegbeli **ThreeDView** ablakot. Ezt eltároljuk a *_view* privát adattagban, illetve megkapja a modellünket, ami segítségével számítja a felületre rajzolandó vásznat. Egyetlen eljárása az osztálynak a **DrawFunction** nevet viseli. Ennek futása alatt a kurzort töltő ikonra állítjuk. Lekérdezzük a képet, majd a *_view*-hoz tartozó canvashoz adjuk. Ezután felcímkézzük a koordinátatengelyeket, Majd a Z tengelyen lévő skálát is felcímkézzük. A tengelycímkéknél félkövér stílust használunk a jobb láthatóság érdekében.

ColorScaleViewModel osztály

A **ColorScaleViewModel** osztályt összekapcsoljuk a **ColorScaleWindow**-val ezt a nézet *DataContext* adattagjával tesszük, hozzákötjük a nézet-modellünket. Példányosításkor

a modellünket kapjuk, amelyet ezután a *_model* privát adattagban tárolunk el. Szükségünk lesz továbbá színek listájára *Colors* néven, amit majd a táblázathoz használunk. *SelectedColor* egy publikus adattag, ami a táblázatban kiválasztott elemnek felel meg. Változáskor *OnPropertyChanged* eljárást hívjuk meg. **LoadData** eljárás végig iterál a modellünk **colorize** adattagján, amely a színeket tárolja. Ezeket belepakolja a **Colors** jelenlegi listába, ami után szintén kiváltjuk az *OnPropertyChanged* metódust. Amennyiben nem sikerült az eljárást végrehajtani, az **OnMessageApplication** ViewModelBase osztálybeli metódussal üzenjük meg a felhasználónak, hogy „Váratlan hiba történt!".

A módosító metódusok találhatók az osztály alján, amelyeket az ablakon lévő gombok megnyomásakor kiváltódó *DelagateCommand*-ok segítségével érünk el. **Modify** eljárás a színmódosításkor hívandó metódus. Színek választásánál nagy segítség a **ColorDialog** osztály aminek egy példányát használjuk, a színválasztáshoz. Amennyiben a szín már korábban létezett a skálában, nem módosítjuk azt, ellenkező esetben pedig a kiválasztott elemet változtatjuk meg. Az **Add** eljárás az előzővel azonos színválasztó ablakban kiválasztott szín ellenőrzése után, ha az nem szerepel a listában, beszúrja a végére. A **Delete** metódus lefutáskor ellenőrzi hogy a jelenlegi listában több elem van-e mint 2. Amennyiben nem a metódus nem fut végig. Ha több elem található abban az esetben a kijelölt elemet eltávolítja a listából.

RBFViewModel

Az **RBFviewModel** osztályt **MainWindow** ablakkal kapcsoljuk össze. A **ViewModelBase** osztályból származtatunk. A konstruktor 2 bemenő paraméterrel rendelkezik. Ebben inicializáljuk a két paramétert, eltároljuk a *_view* és *_model* privát adattagokban. A **DelegateCommand** típusú vezérlőinket, melyeket a konstruktor előtt deklaráltunk, szintén itt csatoljuk az adattagokhoz. Ezeket négy részre lehet bontani. Az elsőbe tartozik a számítási típusokat kiválasztó parancsok. Ezek körébe sorolható a függvényfolytonosság, az epszilon értékének megválasztása, a bázisfüggvény típusa, és a lineáris egyenletrendszer megoldó módszer választása.

A menüpontok előtti kijelölés egyszerű eltávolításához használjuk a **BfTypeUnchecked**, **FTypeUnchecked**, **ETypeUnchecked**, **EUnchecked** metódusokat.

A **SelectedPoint** kiválasztott pontot színezteti a modellel szürkére. Amennyiben van pont kiválasztva, a nézet jobboldalára írja ki annak adatait. Ellenkező esetben nem ír ki adatot. A **RefreshMap** eljárás paraméter nélküli, és a nézet fővásznát, illetve a hozzá tartozó skálát frissíti, azokat címkéire írja a skálát. Az **askBox** metódusban kérdezzük meg a felhasználót arról hogy mit kíván tenni új térkép kezdése előtt. A három lehetséges esetet szétválasztjuk. Az „Igen" esetén mentjük a pontokat, „nem" választásánál viszont

csak egyszerűen új vásznat készít a metódus.

Az eliminációs módszerek cseréjéhez a **ChangeFT**, a bázisfüggvény választásnál a **ChangeType**, függvényfolytonosságnál a **CAB** segéd metódust használjuk.

A pontok beszúrásánál **PointPaste** metódus hívódik meg. Ekkor a vásznon vett pozíciónk helyére leszúrja a pontot a program. A Pontnak törlését a **PointRemove** metódus végzi. Amint sikerült eltávolítania a pontot, abban az esetben frissíti a térképet. A színskála megválasztásánál a **SetColors** metódusba nyitjuk megy a **ColorScaleWindow**-ot és adjuk a `DataContext`-jének a **ColorScaleViewModel** osztályt.

A mentések körébe tartozik a **SaveImage** eljárás, amely megkérdezi hova mentse a `.png` kiterjesztésű fájlt. A pontok mentésére szolgál a **SaveData** eljárás, azoknak betöltésére pedig a **LoadData** metódus. A színskálák mentésére a **SaveColorData**, illetve a **LoadColorData** metódusok szolgálnak.

4.3. Tesztelés

Az alábbi alfejezetben a program helyes futásáról győződhetünk meg. Különválasztjuk a fehér-doboz és fekete-doboz tesztelést. Mindkét tesztelési formának megvan az előnye. A fehér-doboz teszteléssel a modellünk algoritmusainak működését vizsgáljuk, míg a feketével azt nézzük hogy a tervezésben leírtak alapján cselekszik-e a programunk a használat során.

4.3.1. Fehér-doboz tesztelés

Ebben a fejezetben azt fogjuk vizsgálni, hogy a modell megfelelően szolgálja ki a magasabb rétegeinket. A pontokkal foglalkozó metódusokat egységtesztben vizsgáljuk, az eliminációs algoritmusokat pedig összehasonlítás alapján láthatjuk.

Egységteszt

Az egységteszt a program mellett található külön projektben. A projekt egyetlen osztályának konstruktorában példányosítjuk a modellünket, és a tesztosztály eljárásaiban A pontok kezelésével foglalkozó tesztek megadott véletlenszerű értékekre ellenőriz. A program tesztelése során kritikus értékekkel nem kell számolnunk, a modellnek teljesítenie kell a számítást sokkal nagyobb vásznakra, mint ami egy nagy felbontású monitor felülete.

Ponthozzáadás teszt: A modellünk **AddPoint** metódusának tesztjét az azonos elne-

vezésű eljárásban vizsgáljuk.

- 1. ESET : Hozzáadunk pontot a metódushoz, ekkor ellenőrizzük hogy a lista hosszát.
- 2. ESET : A pontunk közelébe lehelyeznénk még egy pontot. Ekkor nem bővülhet a lista mert túl közel van a két pont egymáshoz.
- 3. ESET : Egy megfelelő távolságban lévő pont lehelyezésénél a lista bővül eggyel.

Ponttörlés teszt: A modell **RemovePoint** működését nevével megegyező metódusban tesztelünk le.

- 1. ESET : Nincs kijelölve pont, ekkor próbálunk törölni. Ebben az esetben nem történik semmi.
- 2. ESET : Kijelölünk egy pontot majd azt töröljük. Ekkor a pontok listája csökken eggyel.
- 3. ESET : A pontkijelölésnek nem szabad eltávolított pontot jelölnie.

Értéklekérdezés teszt: A **Value** értéklekérdező metódust számítását a **PointValue** osztályban teszteljük.

- Hozzáadunk pontokat a modellhez, majd azokon vett közelítőfüggvény értékét vizsgáljuk, hogy megegyezik-e a pontoknak megadott értékekkel.

Értéknövelés teszt: A **PointIncrease** helyességét, a **PointIncreasing** eljárásban vizsgáljuk.

- Hozzáadott pontoknál egyik pont értékét növeljük, majd megnézzük, hogy értéke eggyel nagyobb, mint a növelés előtti.

Értékcsökkentés teszt: A **PointDecrease** helyességét, a **PointDecreasing** eljárásban vizsgáljuk.

- Hozzáadott pontoknál egyik pont értékét csökkentjük, majd megnézzük, hogy értéke eggyel kisebb, mint a növelés előtti.

Értékmegadás teszt: A pont értékének változtatását a **PointValue** függvénnyel érhetjük el. Ennek tesztelését a **PointValueChoose** eljárásban végezzük.

- Hozzáadott pontoknál egyik pont értékét megváltoztatjuk, majd megnézzük, hogy értéke megegyezik-e a kívánt értékkel.

Összehasonlító tesztelés

Az összehasonlítást online számítások és a saját programom számítása közt végezzük. Mindhárom saját felbontásra ugyanaz az eredmény jött ki, így nem különböztetem meg a módszereket tesztelésnél. Az online programban [10] ismertsége miatt meg lehet bízni.

A pontokat véletlenszerűen vettem fel a vásznon, azokhoz tartozó fí értékek kerülnek a megoldandó egyenletrendszer mátrixába. A programot Debug módban fogom meg a lépéseknél ekkor a bemenő adatokat látjuk.

A	{double[3, 3]}
[0, 0]	1
[0, 1]	0.27928430546543315
[0, 2]	0.0705647183132795
[1, 0]	0.27928430546543315
[1, 1]	1
[1, 2]	0.02597553877895574
[2, 0]	0.0705647183132795
[2, 1]	0.02597553877895574
[2, 2]	1
b	{double[3]}
[0]	15
[1]	0
[2]	-10
x	{double[3]}
[0]	17.0301848107827
[1]	-4.468307254546489
[2]	-11.08566350562921

4.5. ábra. Tesztértékek a lineáris egyenletrendszer megoldásánál

Az adatokat manuálisan beírva a webes alkalmazásba a következő eredményt kapjuk.

solve

$$x + 0.27928430546543315\,y + 0.0705647183132795\,z = 15$$

$$0.27928430546543315\,x + y + 0.02597553877895574\,z = 0$$

$$0.0705647183132795\,x + 0.02597553877895574\,y + z = -10$$

True

Result:

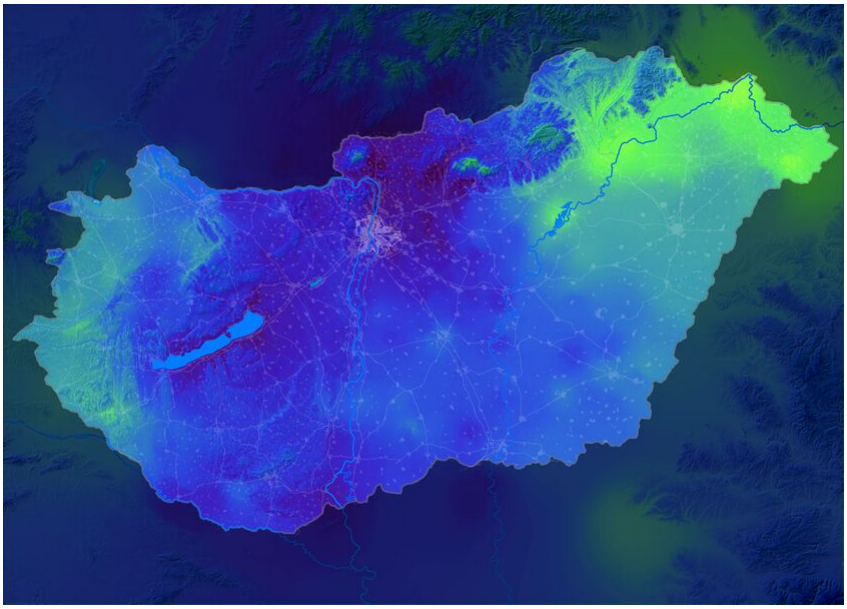
$$x = 17.0301848107827 \text{ and } y = -4.46830725454649 \text{ and } z = -11.0856635056292$$

4.6. ábra. Tesztértékek számítása webes felületen

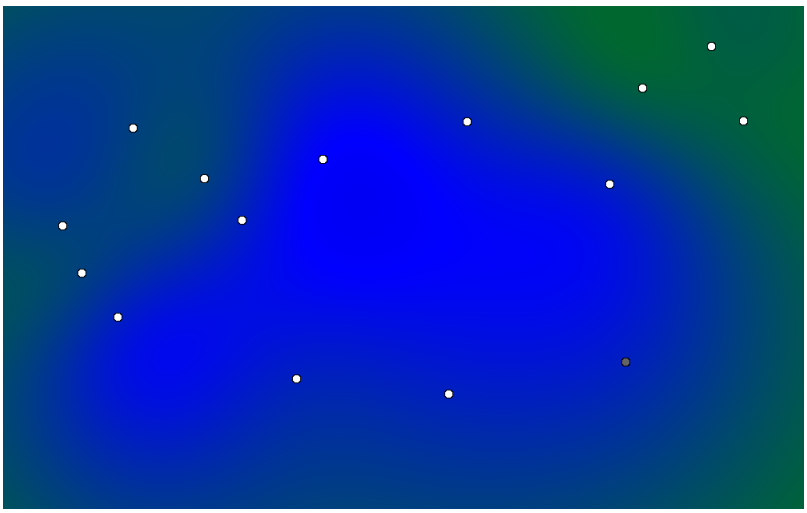
Az összehasonlításban látható hogy nincs eltérés, sőt a miénk egy tizedesjeggyel pontosabb számot adott.

4.3.2. Fekete-doboz tesztelés

Fekete-doboz teszteléssel nem konkrét algoritmusokat tesztelünk, hanem hogy a program helyes működést biztosít-e a felhasználónak. Ehhez ismételten már létező hőtérképeket veszünk alapul. Az időkép [11] oldalán található mai nap hőmérsékleti térképét hasonlítjuk össze a programunk által generált hőtérképpel.



4.7. ábra. Magyarország hőtérképe 2019.12.04-én



4.8. ábra. Magyarország hőtérképe saját közelítésünk szerint

Látható a két ábrán hogy kis eltérés van a színskálázásnál, de a színátmenetek hasonlóak

a két kép között. A lehelyezett pontok mennyisége elegendő a jó közelítéshez, viszont mivel sokkal kevesebb felvett pontunk van, így az átmenetek nem olyan élesek mint az Időkép hőtérképén.

Továbbfejlesztési lehetőségek megvalósítását bármely olvasónak ajánlom, és esetleges megkeresését várom, amennyiben bármi kérdése felmerül a programmal kapcsolatban.

5. fejezet

Befejezés

Továbbfejlesztési lehetőségek

Továbbfejlesztési lehetőségek több irányúak lehetnek. Elképzelésem szerint az alábbiak fejlesztésével lehetne hasznosítani a későbbiekben a programot.

- A programhoz egy segédprogram készítése, mely valós szenzorokkal lenne képes dolgozni, azokat a nekünk szükséges fájlba elhelyezve importálás után valós adatokkal való számítást könnyíthetjük meg vele.
- A program adott tárgyakat megfigyelő hőszenzorok segítségével a tárgyak hőképét vizsgálhatjuk, tehát továbbfejleszthető nem csak domborzati helyek mérésére szolgáló hőkép készítő programmá.
- A program adatbázisban való adattárolása, szintén fejlesztési lehetőség.

Összegzés

Megismerkedhettünk szakdolgozatomban egy interpolációs eljárással, és annak fontosságával. Létrehoztam egy programot, melynek segítségével könnyen szemléltethető. Egyszerű kezelőfelület bemutatásával segítséget nyújtottunk a felhasználóknak, a programháttér leírásával pedig a fejlesztőknek. Továbbfejlesztés esetén szívesen fogadom a fejlesztő megkeresését.

Irodalomjegyzék

[1] Radiális bázisfüggvények interpolációs mátrixa:
https://en.wikipedia.org/wiki/Radial_basis_function_interpolation
2019. december 4.

[2] Radiális bázisfüggvényről:
https://en.wikipedia.org/wiki/Radial_basis_function
2019. december 4.

[3] Epszilon alakparaméter:
https://en.wikipedia.org/wiki/Radial_basis_function#/media/File:Gaussian_function_shape_parameter.png
2019. december 4.

[4] Online függvényábrázoló program:
https://www.desmos.com/calculator
2019. december 4.

[5] Gauss-elimináció:
http://numanal.inf.elte.hu/~krebsz/num12.pdf
2019. december 4.

[6] Cholesky-felbontás:
https://hu.wikipedia.org/wiki/Cholesky-felbont%C3%A1s
2019. december 4.

[7] .NET Framework rendszerkövetelménye:
https://docs.microsoft.com/en-us/dotnet/framework/get-started/system-requirements
2019. december 4.

[8] Windows 7 rendszerkövetelménye:
https://support.microsoft.com/hu-hu/help/10737/

windows-7-system-requirements
2019. december 4.

[9] Gauss-elimináció algoritmusa:
http://www.math-cs.gordon.edu/courses/ma342/handouts/gauss.pdf
2019. december 4.

[10] Wolfram|Alpha:
https://www.wolframalpha.com/calculators/
system-equation-calculator
2019. december 4.

[11] Időkép:
https://www.idokep.hu/hoterkep
2019. december 4.