Állatorvostudományi Egyetem

Bioinformatikai Központ

Földrajzi mintavételezési QGIS-plugin fejlesztése

Készítette:

Papp Márton János

Témavezető:

Dr. Solymosi Norbert

Állatorvostudományi Egyetem, Bioinformatikai Központ,
egyetemi docens

Budapest, 2019

Tartalomjegyzék

1 Rövidítések jegyzéke 2
2 Irodalmi áttekintés 3
2.1 Általános mintavételezés 3
2.2 A reprezentatív mintavételezés 5
2.3 Valószínűségi mintavételezés 6
2.3.1 Egyszerű random mintavétel 7
2.3.2 Szisztematikus random mintavétel 8
2.3.3 Rétegzett random mintavétel: 9
2.3.4 Egyenlőtlen valószínűségi mintavétel: 9
2.4 Mintavételezés térben 10
2.5 Kvadráns rekurzív függvények 12
3 Célkitűzés 13
4 Anyag és Módszer 15
4.1 Felhasznált szoftverek 15
4.2 A módszer ismertetése 16
4.2.1 Térbeli mintavételezés egyenlő kiválasztási valószínűségekkel: 16
4.2.2 Mintavételezés tartalék pontok kijelölésével: 18
4.2.3 Egyenlőtlen valószínűségű mintavétel: 18
5 Az eredmények ismertetése 20
5.1 Egyenlő valószínűségű mintavételezés: 21
5.2 Egyenlőtlen valószínűségű mintavételezés: 26
6 Megbeszélés 33
7 Összefoglalás 36
8 Summary 38
9 Köszönetnyilvánítás 40
10 Irodalomjegyzék 41
11 Melléklet 43

1 Rövidítések jegyzéke

GIS = Geographical Information System,

GRTS = Generlised Random-Tesselation Stratified

MSE = Mean Squared Error

QGIS = Quantum GIS

2 Irodalmi áttekintés

2.1 Általános mintavételezés

A mintavételezés a jelenlegi statisztikai és módszertani eszköztár egyik legalapvetőbb eleme. Számos előnye közül említésre méltó, hogy méretéből adódóan olcsóbb, gyorsabban és könnyebben elemezhető, mint a populáció egészét magában foglaló vizsgálatok (ezt a megközelítést nevezik censusnak). Ugyanezen okokból sokkal korszerűbb és hatékonyabb módszerekkel vizsgálható, amelyek gyakran limitáltan hozzáférhetők (Cochran, 1977).

A jelenlegi széles körű alkalmazása miatt talán hihetetlenül hat, hogy valamikor használatát vita övezte (Kruskal és Mosteller, 1980). A 19. században két statisztikai irányzat bontakozott ki, egyik a vizsgálandó terület teljes lefedését célozta, míg a másik esettanulmányok alapján próbált információt gyűjteni a vizsgált populációról. A minta és a populáció közötti szoros kapcsolatot Anders Nicolai Kiaer norvég statisztikus vetette fel, aki a Nemzetközi Statisztikai Intézet 1895-ös berni ülésén ismertette a reprezentatív mintavételezésre vonatkozó nézeteit, megalkotva ezzel a fogalmat arra a mintára, melyből a populációra vonatkozó ismeretek helyesen becsülhetők (Kruskal és Mosteller, 1980). A vele szemben állók, például Georg von Mayr, azt hangoztatták, hogy a teljes körű vizsgálat eredményeit matematikai módszerek soha nem tudják helyettesíteni, így a reprezentatív minta, bár hasznos, de nem léphet ennek helyébe (Kruskal és Mosteller, 1980). Végül a reprezentatív mintavételezés hívei győzedelmeskedtek, így a kérdés áttevődött ennek a fogalomnak a tisztázására (Kruskal és Mosteller, 1980).

Mintát alapvetően deszkriptív és analitikai célból vehetünk (Cochran, 1977 és Stehman és Overton, 1994). A deszkriptív vagy leíró célból vett mintából a populációnak, illetve a populáció részeinek a jellemzése a cél, csupán ennek alátámasztására végzünk hipotézis vizsgálatot (Stehman és Overton, 1994). Ezzel szemben az analitikai célból vett minta segítségével különbségeket keresünk a populáció eltérő elemei között, vagy azonos elemei között, de eltérő időben. A két cél azonban legtöbbször nem válik el ilyen élesen egymástól (Cochran, 1977). Az orvostudományok sokszor alkalmaznak analitikai célból vett mintát (Cochran, 1977), azonban általában mégis leíró statisztikai vizsgálatok a fő céljai a mintavételezésnek (Stehman és Overton, 1994).

A mintavételezés és annak módszere, habár fontos, de természetesen nem egyedüli eleme annak a folyamatnak, amely során információt szeretnénk kapni, vagy következtetéseket szeretnénk levonni a vizsgált populációról. Ahhoz, hogy a legpontosabb eredményeket kapjuk, a vizsgálat három alapvető elemének összhangban kell lennie. Ezek a populáció, a mintavétel és a mintából végzett számítások (Wang et al., 2010). A populáció milyensége, az elemek közötti kölcsönhatások alapvetően befolyásolják a legideálisabb mintavételi módszert, amely a leginkább kedvez a vizsgálat céljának. A számításoknak pedig összhangban kell lenniük a kezdeti populációval és a mintavétel módszerével is, hogy a lehető legpontosabb eredményt kapjuk (Wang et al., 2010). Cochran (1977) alapján összefoglalható hat lépés, amely az ideális mintavételezési folyamatot jellemzi (Wang et al., 2012). Első lépés a célok leírása, majd következik a populáció, illetve annak elemeinek meghatározása. Mielőtt a pontos mintát kijelölnénk meg kell szabni a mintaelemszámot és a módszert, majd következik a mintavételezési terv elkészítése. Ebben meg kell határozni a mintavételezési helyeket, a mintázandó elemeket és a mintavételek pontos időpontját. Végül végre kell hajtani a mintavételezést és el kell végezni az adatok elemzését.

A mintavétel módszerétől és a mintától általában elvárás, hogy reprezentatív legyen, azonban ez a fogalom a mai napig nem teljesen tisztázott (Olsen et al., 2012). A leggyakrabban alkalmazott módszer reprezentatív minta nyerésére, a valószínűségi mintavétel (Olsen et al., 2012). A következő fejezetekben ezért ennek a módszernek a tárgyalása következik.

A valószínűségi mintavétellel szemben állnak azok a módszerek, amelyek a véletlenszerűség helyett a gondos, szakértő tervezést helyezik előtérbe. Ide tartoznak a célzott mintavétel különféle típusai. Ezek során olyan elemeket keres a mintavételező, amelyek valamilyen szempontból tipikusak a populációra (Cochran, 1977). Másik gyakran alkalmazott módszer a kényelmi mintavétel, ahol azok az elemek kerülnek a mintába, melyek a mintavételezés időpontjában a legkényelmesebben elérhetők. A fenti módszerek pontossága, ha ismertek a populáció vonatkozó adatai, vagy azokat valószínűségi mintavétellel már meghatároztuk, elbírálható. Időnként pontos eredményhez is juthatunk általuk, azonban ez nem történik meg minden helyzetben. Másik nagy hátrányuk, hogy a mintából nem becsülhető a számítások hibája (Cochran, 1977). Ezeket olyankor alkalmazzák elsősorban, amikor nem cél a populációról pontos következtetéseket levonni, illetve, amikor korlátozott anyagi források állnak rendelkezése (Etikan et al., 2015).

2.2 A reprezentatív mintavételezés

Ma szinte minden mintavétel célja a reprezentativitás, azonban a reprezentatív mintavétel, mint módszer sokáig vita tárgya volt (Kruskal és Mosteller, 1980), sőt még ma sem tisztázott teljesen a fogalom (Olsen et al., 2012).

A Nemzetközi Statisztikai Intézet 1895-ös ülésén, amikor Kiaer megfogalmazta nézeteit a reprezentatív módszerrel kapcsolatban, a statisztikusok többsége még a mintavétel helyett a vizsgált populáció teljes lefedését tartotta kívánatosnak. Ez elsősorban a szociológiai tudományokra volt jellemző, hiszen a természettudományok területén már korán felismerték, hogy az ilyen törekvések nagyon nehezen megvalósíthatók (Kruskal és Mosteller, 1980).

Kiaer reprezentatív módszere még nem valószínűségen alapult. Egy olyan módszert ismertetett, amelyben szisztematikus, de nem valószínűségi alapokon, egyenlően elosztva történik a mintavétel. Az egyenlő elosztás elsősorban földrajzi értelemben jelent meg, de amennyiben a populáció egyes elemeiből kevesebb volt a mintában, akkor azok pótlását is ide értette. A reprezentatív mintát a populáció miniatűrjének tekintette (Kruskal és Mosteller, 1980).

A Nemzetközi Statisztikai Intézet üléseinek során a valószínűség és a mintavétel kapcsolatának első komoly megfogalmazása von Bortkiewicz-től származik (Kruskal és Mosteller, 1980). Poisson módszerét felhasználva, mely azt mutatja meg, hogy két arány különbsége mennyiben tekinthető véletlenszerűnek, próbálta tesztelni Kiaer módszerének reprezentativitását. Úgy vélte, hogy amennyiben a vizsgált változó különbsége a minta és a populáció esetén a véletlenszerűség határán túl van, akkor a minta nem tekinthető reprezentatívnak (Kruskal és Mosteller, 1980).

1903 és 1925 között kevés szó esett a mintavételezésről, azonban ki kell emelni Bowley-t, aki 1915-ben négy célzottan kiválasztott angol városban alkalmazott szisztematikus mintavételt és eredményei pontosságát is meghatározta, méghozzá átlag +- 3-szor a valószínű hiba formájában. Említésre méltó ebből az időből Chuprov is, aki 1923-ban közölt írásában már hangsúlyozta a valószínűségen alapuló módszer jelentőségét (Kruskal és Mosteller, 1980).

Végül a Nemzetközi Statisztikai Intézet 1925-ös római ülésén már az új generáció győzedelmeskedett (Kruskal és Mosteller, 1980). A mintavételezés elfogadott módszerré vált, így innentől kezdve a módszerek és az adatok elemzése került a vita középpontjába (Kruskal és Mosteller, 1980).

A reprezentatív minta és reprezentatív mintavétel kifejezések a mai napig változó jelentéssel bírnak (Olsen et al., 2012). Kruskal és Mosteller (1979) a statisztikai irodalomban összegyűjtötte a fogalomhoz tartozó különféle jelentéseket. Ebben az írásban megjelenik, mint általános adatgyűjtési módszer, olyan minta, ami szelektív hatásoktól mentes, a populáció miniatűrje, tipikus a populációra, a precízió értékmérője, egy specifikus módszer, jó közelítés és olyan minta mely egy adott célra megfelelő. Olsen és munkatársai (2012) ezek közül a reprezentatív mintát, mint a populáció miniatűrjét határozták meg, melyhez reprezentatív módszerrel jutottunk. A valószínűségi mintavételezés olyan módszer, amely alkalmas reprezentatív minta előállítására, hiszen amennyiben egyenlő minden elem kiválasztási valószínűsége, mentes a szelektív hatásoktól (Stephan és McCarthy, 1958). Azonban az ilyen módszerek is eredményezhetnek olyan mintát, amely nem reprezentatív (Kruskal és Mosteller, 1979 és Olsen et al., 2012), ezért a populáció elemei közti kapcsolatra különösen fontos odafigyelni a minta elemei között (Olsen et al., 2012). A valószínűségi mintavétel előnye, hogy a becslések pontossága meghatározható (Cochran, 1977), amely fontos a minta reprezentativitásának megítélésében, hiszen az ilyen minta célja, hogy biztos alapot adjon a populációra vonatkozó becsléseknek (Stephan és McCarthy, 1958).

2.3 Valószínűségi mintavételezés

Legyen *U* egy véges populáció, melynek *N* eleme van, vagyis $\{U = u_1, u_2, u_3, \ldots, u_N\}$. Ha a populáció egy tulajdonsága *Y*, akkor a tulajdonságnak a populáció egyes elemeihez tartozó értékei $y_1, y_2, \ldots, y_N$ (Sampath, 2001 és Fuller, 2009). Az *U* populációból kiválasztható *n* elemű minták legyenek $s_1, s_2, \ldots, s_u$. Minden *s* lehetséges minta eleme *S* populációnak (Cochran, 1977 és Sampath, 2001). A lehetséges *n* elemű minták, vagyis a *S* populáció elemeinek száma

$$C_N^n = \binom{N}{n} = \frac{N!}{n!(N-n)!}$$

(Horvitz és Thompson, 1952 és Fuller, 2009).

A valószínűségi minta olyan minta, ahol minden lehetséges *s* minta és a hozzá tartozó *p(s)* valószínűség ismert és nullánál nagyobb (Stehman és Overton, 1994). Ekkor felírható, hogy

$$\sum_{s \in S} p(s) = 1$$

(Stehman és Overton, 1994).

Annak ellenére, hogy a lehetséges minták és a hozzájuk tartozó valószínűségek ismertek, nem szükséges felírni őket a mintavételezés során, elegendő tudni, hogy felírhatók (Cochran, 1977).

Ha a fenti *p(s)* valószínűségi eloszlás és az összes u_i-t tartalmazó $S^{(i)}$ minta száma ismert, akkor az *U* populáció egy *i* elemének teljes mintavételre vonatkozó kiválasztási valószínűsége *n* elemű minta esetén

$$P(ui) = \sum_{s}^{S^{(i)}} P(s),$$

ahol $S^{(i)}$ azoknak a mintáknak a száma, amelyek tartalmazzák u_i-t (Horvitz és Thompson, 1951).

A minta alapján lehetőségünk van becslést adni a populáció értékeire. Ha a becslés nem torzított és ismert a változó átlaga és szórása, akkor a normál eloszlás alapján meghatározhatók a hozzá tartozó különféle konfidencia intervallumok (Cochran, 1977).

A becslés torzítása két okból következhet be. Az egyik, hogy a becsléshez használt módszer torzít, míg a másik, hogy a mintavétel és a mérés során következtek be hibák, pontatlanságok (Cochran, 1977). A torzítás magával vonja a becsült szórás és következésképpen a konfidencia intervallum torzulását is.

T paraméter becslése a *s* mintából legyen t_s. Ha $E(t_s) = T$, akkor a torzítás 0, vagyis a becslés mentes a torzítástól (Sampath, 2001). A torzítás mértékét az $E(ts)-T$ különbség adja meg (Stehman és Overton, 1994).

Két eltérő mértékben torzított, vagy egy torzított és egy nem torzított becslés összehasonlítására alkalmas mérőszám az átlagos négyzetes hiba (mean squared error, a továbbiakban MSE).

$$MSE = E(t_s-T)^2$$

(Cochran, 1977). Bizonyítható, hogy MSE egyenlő t_s varianciájának és a torzítás négyzetének összegével, ezért alkalmas két minta torzításának összehasonlítására (Cochran, 1977).

2.3.1 Egyszerű random mintavétel

Az egyszerű random mintavétel olyan valószínűségi módszer, ahol az összes s $(s \in S)$ minta ugyanakkora valószínűséggel kerül kiválasztásra (Cochran, 1977). Ekkor annak a valószínűsége, hogy egy mintát kiválasztunk a következők szerint alakul: annak a

valószínűsége, hogy az *n* elemű minta első eleme kiválasztásra kerül $\frac{n}{N}$. A következőnek $\frac{(n-1)}{(N-1)}$. Ha ezt folytatjuk az utolsó elemig, akkor ezek szorzataként megkapjuk az adott *s* minta kiválasztási *p(s)* valószínűségét (a fenti csak visszatevés nélküli mintavételezés esetén igaz). Ebből következik, hogy

$$\frac{n}{N} \cdot \frac{(n-1)}{(N-1)} \cdot \frac{(n-2)}{(N-2)} \cdot \ldots \cdot \frac{1}{(N-n+1)} = \frac{1}{C_N^n}$$

ahol C_N^n az összes lehetséges minta száma (Cochran, 1977).

Egy *ui* elem kiválasztási valószínűsége felírható

$$p(u_i) = \sum_{s \in S^{(i)}} p(s),$$

ahol $S^{(i)}$ az összes olyan minta aminek u_i eleme (Stehman és Overton, 1994).

A minta átlaga

$$\bar{y} = \sum^{n} \frac{y_i}{n},$$

ahol *i = 1, 2, ..., n,* alapján becsülhető a $\bar{Y}$ populáció átlag (Cochran, 1977).

2.3.2 Szisztematikus random mintavétel

A szisztematikus mintavétel során az *N* elemű populáció elemeit olyan módon mintázzuk, hogy az első random kijelölt elem után, minden *k*-adik elemet a mintába teszünk, $k = \frac{N}{n}$. Előnye elsősorban a mintavételezés gyakorlati megvalósításakor mutatkozik, hiszen sok esetben könnyebben kivitelezhető, mint a többi módszer, azonban, mivel az egyes pontok térben egyenletesen vannak elosztva, térbeli mintavételezés során az egyszerű random mintavételnél pontosabb (Cochran, 1977).

Ezzel szemben a szisztematikus random mintavételnél, ahol populáció elemei random sorrendbe vannak rendezve és ebből történik a mintavétel, az egyszerű random mintavétellel közel ekvivalensnek tekinthető (Cochran, 1977). Ebben az esetben a szisztematikus random mintavétel és az egyszerű random mintavétel varianciája átlagosan megegyezik, azonban a szisztematikus mintavétel varianciája kissé szabálytalanabbul változik mintánként, ha kicsi a mintaelemszám (Cochran, 1977).

2.3.3 *Rétegzett random mintavétel:*

Rétegzett mintavétel során a populációt bizonyos számú elkülönült rétegere bontjuk. Ha L a rétegek száma, akkor az összes elem $N = N_1 + N_2 + \ldots + N_L$, ahol N a populáció elemeinek száma (Cochran, 1977). Amennyiben a rétegeken belül a minta random kerül kiválasztásra, rétegelt random mintavételről beszélünk (Cochran, 1977).

Erre a módszerre akkor lehet szükség, ha a populáció egyes elemeit önállóan érdemes kezelni, ezek becsült értékeire külön van szükség, illetve, ha gyakorlati szempontok alapján ez indokolt (Stehman és Overton, 1994).

Rétegelt mintavétel esetén u_i elem kiválasztási valószínűsége

$$p(u_i) = \frac{n_h}{N_h},$$

ahol n_h a h rétegben vett minták száma és N_h az összes elem a h rétegnek (Stehman és Overton, 1994).

Rétegelt mintavétel során a populáció átlag becslése

$$\bar{y}_{st} = \sum_{h=1}^{L} \frac{N_h \bar{y}_h}{N} = \sum_{h=1}^{L} W_h \bar{y}_h,$$

ahol $W_h = \frac{N_h}{N}$ *és* $\bar{y}_h = \sum_{i=1}^{n_h} \frac{y_{hi}}{n_h}$ (Cochran, 1977).

Amennyiben $\frac{n_h}{n} = \frac{N_h}{N}$, akkor a populáció átlag megegyezik a mintákból számítható átlaggal, vagyis $\bar{y}_{st} = \bar{y} = \sum_{h=1}^{L} \frac{n_h \bar{y}_h}{n}$ (Cochran, 1977).

2.3.4 *Egyenlőtlen valószínűségi mintavétel:*

Bizonyos esetekben kívánatos, hogy a populáció elemei ne egyforma valószínűséggel kerüljenek a mintába. Erre lehet szükség, amikor egyes elemeknek alacsony az előfordulása, de jelenlétük a mintában jelentős a vizsgálat szempontjából. Véges populációk esetén Horvitz és Thompson (1952) becslést határozott meg egyenlőtlen mintavételezés esetére.

Ekkor például a mintaelemek egy tulajdonságának várható értéke a következő formában írható fel, a populáció elemeinek kiválasztási valószínűségei alapján

$$E(\sum_{i=1}^{n} y_i) = \sum_{i=1}^{N} p(s_i) Y_i,$$

ahol Y_i a vizsgált változó valós értéke (Horvitz és Thompson, 1952).

Általánosan formában Y pontos becslése felírható

$$\hat{Y} = \sum_{i=1}^{n} \beta_i y_i$$

formában, ahol β_i az i-edik elemhez tartozó súly, amely korrigálja az egyenetlen kiválasztási valószínűségből következő eltéréseket (Horvitz és Thompson, 1952).

Amennyiben egyenlő minden elem kiválasztási valószínűsége akkor

$$\hat{Y} = \frac{N}{n} \sum_{i=1}^{n} y_i.$$

Egyenlőtlen valószínűségek esetén, ahhoz, hogy a becslés torzítástól mentes legyen, igaznak kell lennie, hogy

$$\sum_{i=1}^{N} p(s_i) \beta_i Y_i = \sum_{i=1}^{N} Y_i.$$

Ahhoz, hogy az előbbi egyenlet igaz legyen, teljesülnie kell a következőnek, $\beta_i p(s_i) = 1$. Ebből következik, hogy

$$\hat{Y} = \sum_{i=1}^{n} \frac{y_i}{p(s_i)},$$

melyet Horwitz-Thompson becslésnek nevezünk (Horvitz és Thompson, 1952).

Cordy (1993) meghatározta az egyenlőtlen kiválasztási valószínűségű mintavételhez tartozó becslést végtelen populációk esetében. A fenti esetben Z változó összegének becslése

$$\hat{Z} = \sum_{i=1}^{n} \frac{z(s_i)}{\pi(s_i)},$$

ahol n a mintaelemszám, $z()$ az a függvény, amely meghatározza a vizsgált a változót, aminek az összegét becsülni szeretnénk, míg $\pi(s)$ a kiválasztási sűrűségfüggvény.

A kiválasztási sűrűségfüggvény megadásához ki kell jelölni n random változót U populáción, ami $S = \{s_1, s_2, \dots, s_i\}$ lesz. Ezek kapcsolt valószínűségi sűrűségfüggvénye $f(s_1, s_2, \dots, s_i)$, ahol s_i-hez tartozó marginális valószínűségi sűrűségfüggvény $f_i(s)$. Ekkor $x \in U$ esetén

$$\pi(x) = \sum_{i=1}^{n} f_i(x),$$

ahol $\pi(x)$ a kiválasztási sűrűségfüggvény, amely megmutatja, hogy egy területen mekkora a kiválasztásra kerülő mintaelemek száma (Cordy, 1993 és Stevens, 1997)

2.4 Mintavételezés térben

A fenti mintavételezési módszerek alkalmazhatók térbeli vizsgálatok során is. Lehetőség van egyszerű random mintavételre, ahol a random pontokat a térben jelöljük ki.

Szisztematikus mintavétel során szögletes hálót helyezünk random a populációra, míg rétegelt mintavétel során a populációt részekre osztjuk, majd ezekben valamilyen módszerrel kijelöljük a mintaelemeket (Stehman és Overton, 1994).

Térbeli mintavételezés során a populáció elemeinek függetlensége nem teljesül (Wang et al., 2012). A térbeli autokorreláció következményeként megfigyelhető, hogy az egymáshoz térben közelebb elhelyezkedő elemek nagyobb valószínűséggel lesznek egymáshoz hasonlók, mint a távolabb lévők (Wang et al., 2012). Gyakorlatilag ahogy a térbeli autkorreláció növekszik a mintában, úgy a megduplázódott információtartalom is vele növekszik (Griffith, 2005). Mérsékelheti az autokorreláció hatását, ha a mintaelemeket térben egyenletesen helyezzük el, ezért a szisztematikus mintavétel alkalmazása ezekben az esetekben hatékonyabbnak tekinthető (Wang et al., 2012 és Griffith, 2005).

A térbeli populációkra jellemző heterogenitás tovább növelheti a mintából történt becslések és összehasonlítások pontatlanságát (Wang et al., 2012). Ennek leküzdésére alkalmazható olyan eljárás, ahol a teret homogén részekre osztjuk, majd ezekből történik a mintavétel (Wang et al., 2010). A fenti két szempontot azonban gyakran nehéz összeegyeztetni, hiszen a populációelemek elhelyezkedése sokszor nem olyan homogén, mint amit a mintaelemek térbeli kiegyensúlyozásánál elvárunk (Wang et al., 2010 és Griffith, 2005).

A térbeli egyensúly vizsgálatára alkalmas módszer például a Voronoj cellák használata. Ekkor egy cella azt a területet foglalja magában, ahol a populáció elemei közelebb vannak egy adott mintaelemhez, mint bármely másikhoz. A minta akkor van egyensúlyban, ha egy cellán belül a kiválasztási valószínűségek összege 1 (Grafstörm és Schelin, 2013).

Olsen és munkatársai (2012) a későbbiekben tárgyalt Generalised Random-Tesselation Stratified (Általánosított Random-Mozaik Rétegzett, továbbiakban GRTS) módszer kapcsán a térbeli kegyensúlyozottságot három teszttel vizsgálta, melyek Voronoj polygonokon-, méghozzá a polygon és a teljes mintázandó réteg arányán alapultak, melyet p_i-vel jelölünk. Ideális térbeli eloszlás esetén $p_i = 1/n$, ahol n a mintaelemek száma. Egyik teszt a Pielou index, amit felírhatunk $J_p = H/H_{max}$ formában, ahol H a Shannon-Wiener diverzitás index, míg H_{max} ennek a maximális értéke. Ideális esetben $J_p = 1$. A két további a X_p^2 teszt, ami $X_p^2 = \sum_{i=1}^{n} \frac{(p_i - \frac{1}{n})^2}{(\frac{1}{n})}$, illetve a térbeli egyensúlyi mérték, mely $S_p = \sqrt{\sum_{i=1}^{n} \frac{(p-\bar{p})^2}{n-1}}$. Ezek értéke ideális esetben nulla.

2.5 Kvadráns rekurzív függvények

Stevens és Olsen (2004) módszerének alapja, hogy a kétdimenziós teret egydimenzióssá alakítja. Az egydimenziós térből szisztematikus minta vehető. Ezt olyan módon teszi meg, hogy közben a térbeli viszonyok a lehető legjobban megőrződjenek, vagyis a populáció térbeli szempontok szerinti mintázása ilyen formában is lehetségessé váljon. Ehhez kvadráns rekurzív függvényeket alkalmaznak (Stevens és Olsen, 2004).

A kvadráns-rekurzív módszer során a teret $2^m \times 2^m$ négyzetes mátrixxá alakítjuk, ahol *m* nullánál nagyobb valós szám (Mark, 1990). Ezután a mátrixot visszafele haladva egyre magasabb szintű kvadránsokra osztjuk, miközben a cellákhoz egy sorszámot rendelünk (Mark, 1990). Ezt céltól függően több módon is meg lehet tenni. A cellákat meghatározott rend szerint is sorszámozhatjuk, azonban azokat 4 sorszám random permutációjaként is elnevezhetjük (Mark, 1990). Az egyes szintekhez tartozó név adja meg az elem helyét a térben.

A kvadráns rekurzív módszer képes többé-kevésbé megőrizni a térbeli kapcsolatokat, ezért alkalmas például térkép adatbázisok szerkesztésére, hiszen a térben egymáshoz közel elhelyezkedő elemek az adattárolóban is egymáshoz közel helyezkednek el (Mark, 1990). Stevens és Olsen (2004) módszerében random permutációval sorba rendezve, kvadráns rekurzív függvénnyel hierarchikusan randomizálják a populáció elemeit, miközben a térbeli kapcsolatok többé-kevésbé megmaradnak.

3 Célkitűzés

A mintavételezés a tudományos kutatás egyik alapvető adatgyűjtési módszere, meghatározó a természettudományos és ezzel együtt a biológiai tudományok területén egyaránt. Ennek alapja az olyan minta, amelyből megbízható becslések és következtetések vonhatók le a vizsgált populáció tekintetében. Az ilyen mintát reprezentatívnak nevezzük. Az a minta, amely a populáció miniatűrje, általában reprezentatívnak tekinthető, hiszen ekkor a populációra jellemző eloszlás jelenik meg a mintában is.

Térbeli mintavételezés során a mintaelemek elhelyezkedése is információtartalommal bír, mivel a közeli elemek feltételezhetően hasonló hatásoknak vannak kitéve, ezért bizonyos tulajdonságaikban hasonlók lesznek. Ennek következtében az egyenletesen elosztott mintapontok általában jobb eredményt adnak.

Stevens és Olsen (2004) módszere, a GRTS a kétdimenziós teret kvadráns rekurzív módszerrel egy egyenesen ábrázolja. Ennek során a térbeli kapcsolatok megőrzése mellett a térbeli elemek lehető legvéletlenszerűbb sorrendjét kapjuk. Az egyenesről vett szisztematikus minta térben kiegyensúlyozott, de véletlenszerű.

Munkánk során az volt a célunk, hogy a GRTS módszer alkalmazását olyan módon készítsük el, hogy az mindenki számára elérhető és könnyen használható legyen, elsősorban az állatorvosi-, epidemiológiai- és más biológiai tudományok területén. Ezért döntöttünk úgy, hogy QGIS (Quantum GIS) Python plugin-ként készítjük el. A plugin egy szoftvert kiegészítő modul, a QGIS programban az új funkciók bevezetése elsősorban felhasználók által fejlesztett pluginokon keresztül történik. A QGIS előnye a felhasználó szempontjából, hogy ingyenes, és a földrajzi információ egyszerűen, grafikus felhasználói felület segítségével kezelhető. Ezáltal az algoritmus nyújtotta előnyöket a kutatástervezés során mindenki élvezheti.

A plugin szerkesztésekor célunk volt, hogy könnyen átlátható és egyszerűen használható legyen, ezért olyan grafikus felületeket terveztünk, melyeken a funkciók nehézségek nélkül kezelhetők, azok működése jól átlátható.

Amennyiben a felhasználó olyan adatokat ad meg, melyek formátuma nem megfelelő, vagy más módon hibát generál, a program a hiba típusáról üzenetben tájékoztatja a felhasználót. Innen ahhoz a legkorábbi részhez lehet visszatérni, amely beállítása során hiba nem jelentkezett, így a program használata zökkenőmentes marad.

Célunk volt, hogy a plugin esetleg jelentkező hibái könnyen nyomonkövethetők, és javíthatók legyenek, mely a QGIS Python plugin könyvtár segítségével egyszerűen megvalósítható. Itt ugyanis megtalálható a plugin hibakövetőjének linkje és egyéb hasznos információk a problémamentes felhasználáshoz.

4 Anyag és Módszer

4.1 Felhasznált szoftverek

Munkánk során elsődleges célunk volt, hogy az elkészített plugin könnyen és ingyenesen hozzáférhető legyen mindenki számára, ezért az elérhető GIS (Geographical Information System, magyarul térinformatikai rendszer) szoftverek közül a Quantum GIS-re (továbbakban QGIS) esett a választásunk.

A GIS programok olyan szoftverek, amelyek földrajzi adatokat képesek kezelni. Általában grafikus felhasználói felülettel rendelkeznek, így egyszerűen vezérelhetők. Segítségükkel lehetőség nyílik digitális térképek készítésére, szerkesztésére, valamint ezek könnyen testre szabható megjelenítésére. A hagyományos térképekkel szemben további előnyük, hogy a térképen megjelenő objektumokról különféle adatokat képesek tárolni, ezeket a QGIS az attribútum táblában kezeli. A táblázat vízszintes sorai képviselik az elemeket, amelyek a térképi réteget alkotják, míg a függőleges oszlopok, a mezők tartalmazzák egy tulajdonság objektumokhoz tartozó értékeit.

A földrajzi információt a GIS programok két alapvető formában kezelik. Vektor rétegek esetén az objektum helyzetét annak X- és Y-, esetleg Z koordinátája adja meg. Egy csúcsként, ún. vertex-ként jelennek meg a pont objektumok, melyek térbeli helyzetét azok koordinátája határozza meg. Vonalakat kettő vagy több vertex szab meg, melyeket folytonos vonal köt össze, míg a terület, vagy polygon jellemzője, hogy három- vagy annál több vertex által körülhatárolt régióként tárolódik és jelenik meg a térképen.

A térképi rétegek másik típusa a raszter réteg, mely esetében a tér cellákra, vagy más néven pixelekre van felosztva, és minden pixelhez egy érték van rendelve. Raszter rétegek segítségével könnyen tárolhatók folytonos változók adatai, például egyes fajok sűrűsége vagy az átlagos csapadék mennyisége. Másik felhasználási területe ezen rétegeknek a régiókról készült légi- vagy műholdfelvételek megjelenítése. Ebben az esetben három egymásra helyezett réteg alkotja a végső képet, mely három réteg egyenként a vörös, zöld és kék színeket tartalmazza.

GIS rendszerek állatorvosi epidemiológiai felhasználására már több hazai példa is megtalálható (Solymosi és Medveczky, 2000, Solymosi et al., 2004 és Solymosi et al., 2010).

Azért esett választásunk a GIS szoftverek közül a QGIS-re, mert ez egyaránt használható Windows-on, macOS-en és a legtöbb UNIX platformon, valamint ingyenes, ezért mindenki számára elérhető. A program nyílt forráskódú és szabadon módosítható, ezért a felhasználók közössége részt vesz a folyamatos fejlesztésekben, frissítésekben, új funkciók bevezetésében. A funkciók testreszabásának, illetve új funkciók létrehozásának legáltalánosabb módja a pluginok fejlesztése. Léteznek ún. core pluginok, melyeket a szoftver fejlesztői tartanak fenn, valamint olyanok, amelyeket a felhasználók készítettek. Pluginok fejlesztése C++ vagy Python nyelven történhet. Munkánk során a Python nyelvre esett a választásunk, mert ezek a QGIS Python Plugin könyvtárba feltöltve egyszerűen megoszthatók, könnyen fejleszthetők és fenntarthatók.
Munkánkat a QGIS 3.8-as verziójához készítettük el, a kódot a Python 3.7.3-as verziójában írtuk. A plugin gerincét a QGIS Plugin Builder nevű QGIS plugin 3.2.1-es verziójával készítettük el. A grafikus felhasználói felületeket a PyQt 5-ös verzója és a Qt Creator 4.7.1-es verziója segítségével hoztuk létre.

4.2 A módszer ismertetése

Stevens és Olsen (2004) módszerének alapja, hogy a kétdimenziós teret egy egyenesre képezi le, miközben a térbeli kapcsolatokat többé-kevésbé megőrzi és a lehető leginkább randomizálja az elemeket. Ebből random pontból induló szisztematikus mintavételezés segítségével történik a mintapontok kiválasztása. A módszert Generalized Random-Tesselation Stratified, rövidítve GRTS modellnek nevezték el, széles körben alkalmazzák, többféle felhasználási területe között van például a légköri, erdészeti, vagy vízi vizsgálatok tervezése (Brown et al., 2015).

4.2.1 Térbeli mintavételezés egyenlő kiválasztási valószínűségekkel:

A térbeli információ egyenesre történő leképezése egy random elhelyezett négyzetrács kvadráns rekurzív felosztásán alapul (Stevens és Olsen, 2004 és Mark, 1990). A földrajzi rétegre egy olyan egyenlő cellanagyságú négyzetrácsot kell helyezni, amely annak legnagyobb kiterjedésén az X és Y tengelyen is egy cella nagyságával nyúlik túl. Ennek segítségével a négyzetháló random helyezhető a kiválasztott térképi rétegre, aminek következtében minden két pont eltérő cellába kerülhet (Olsen et al., 2012). Ahhoz, hogy a térbeli információ kvadráns rekurzív módon leképezhető legyen egy egyenesre, a

négyzetháló méretének $2^m \times 2^m$-nek kell lennie, ahol m nullánál nagyobb egész szám, vagyis az összes cella száma 4^m. A minimálisan szükséges m szint egyenlő $\log_4(n)$-nel, ahol n a mintaelemszám (Olsen et al., 2012).

A felosztás elmélete szerint először négy cellából álló háló kerül felhelyezésre a térképi rétegre, majd ezek egytől négyig terjedő random sorszámot kapnak. Ezután minden cellát további négy cellára osztunk és ezeket ugyanúgy sorszámmal látjuk el, mint az egyel magasabb hierarchikus szint esetében. Ezt addig folytatjuk, amíg az utolsó m-edik szint esetében is sorszámot rendeltünk a cellákhoz. A cellák sorrendjét az így létrehozott sorszámok határozzák meg (Stevens és Olsen, 2004). Ezt a folyamatot hierarchikus randomizálásnak nevezzük. A sorszámok olyan módon kerülnek kialakításra, hogy azok minden helyiértékén az adott hierarchikus szinthez tartozó cella sorszáma található.

Az egyenes hosszúságát a kívánt mintanagyság határozza meg, ezt a cellák számával egyenlő, azonos hosszúságú részre osztjuk. Az egyes szakaszok az egyes celláknak felelnek meg, sorrendjük a fentiek szerint meghatározott (Olsen et al., 2012).

A módszer következő lépésében az egyes cellákhoz tartozó szegmensen ki kell jelölni a cellán belül található polygonoknak, vagy polygon részleteknek megfelelő szakaszokat. A polygon részlethez tartozó szakasz hosszúsága egyenlő a teljes szakasz olyan arányú részével, mint a polygon aránya a cellában. Az egyes polygon részletekhez tartozó szakaszok random kerülnek a cellához tartozó szakaszon belül felosztásra (Olsen et al., 2012).

A fentiek szerint elkészített egyenesről szisztematikus mintavétellel juthatunk a mintaelemekhez. A mintavétel random kezdettel indul, majd pontok kerülnek kijelölésre minden k távolságban, ahol $k = N/n$, n a kívánt minták, míg N a teljes egyenes hosszúsága (Stevens és Olsen, 2004).

A mintavételezési helyek a szisztematikus módon meghatározott pontok szerint kerülnek kijelölésre. Amely cellába és azon belül, amely polygon részlethez tartozó szakaszba esik a kijelölt pont abban a polygon részletben történik a mintavételezési hely random kijelölése (Olsen et al., 2012).

Voronoi polygonok alapján Olsen és munkatársai (2012) összehasonlították a GRTS és az egyszerű random mintavétel térbeli kiegyensúlyozottságát. Ehhez Pielou indexet, a X_p^2-t és a térbeli egyensúlyi mértéket vették figyelembe. Vizsgálatuk során azt találták, hogy az esetek legnagyobb részében a tesztek ideálisabb értéket adtak a GRTS esetében, mint az egyszerű random mintavételnél.

4.2.2 Mintavételezés tartalék pontok kijelölésével:

Bizonyos esetekben az eredetileg meghatározott mintavételezési pontokon lehetetlen elvégezni a mintavételezést. Ennek oka lehet például, hogy az adott pontban nem található meg a vizsgált populációnak egy tagja sem, vagy a terület elérhetetlen, akár az elhelyezkedése miatt, akár azért, mert magánterületre esik, illetve anyagi és időbeli okai is lehetnek elemek kiesésének. Ilyen esetekben hasznos, ha vannak olyan további helyek kijelölve, amelyekből ezen esetekben pontokat emelhetünk be a mintába (Olsen et al., 2012).

Amennyiben a fentiek szerint több mintaelemet jelölünk ki, mint amennyit valóban vizsgálni szeretnénk, akkor a szisztematikus mintavétel következtében üres területek fognak megjelenni. Ezeknek a mintapont nélküli régióknak a nagysága a többletminta és a valódi minta arányával egyenlő (Olsen et al, 2012).

Stevens és Olsen (2004) megoldást javasolt a fenti problémára. Ennek során a cellához tartozó sorszám helyiértékeit megfordítjuk úgy, hogy például a legkisebb helyiérték kerül a legnagyobb helyére. Amennyiben utána eszerint rendezzük sorba a cellákat, akkor a végeredményben továbbra is közel térben egyenletes mintát kapunk. Igaz ez abban az esetben is, ha tartalék pontok is kijelölésre kerülnek (Stevens és Olsen, 2004). Csökkentve a mintaelemszámot azonban a GRTS módszer hatékonysága közelít, majd igen kis számú minta esetén egészen lecsökken az egyszerű random mintavétel hatékonyságának szintjére. (Olsen et al., 2012).

4.2.3 Egyenlőtlen valószínűségű mintavétel:

Amennyiben a vizsgálat szempontjából kívánatos bizonyos területek előnyben részesítése a mintavétel során, egyenlőtlen valószínűségű mintavételt érdemes használni. Stevens és Olsen (2004) erre két lehetőséget ad meg. Az egyik szerint bizonyos területek kiválasztási valószínűségét növeljük, másokét pedig csökkentjük és ezzel párhuzamosan módosítjuk az egyenesen elfoglalt területüket (Olsen et al., 2012). A másik módszer szerint a populációt részekre osztjuk a számunkra fontos tulajdonságok alapján, majd minden részben önállóan elvégezzük a mintavételezést a kívánt mintaelemszámmal. Ezt nevezzük rétegzett mintavételezésnek. Mi, munkánk során ez utóbbi megoldást választottuk, azért, mert ebben az esetben bizonyosan tudható, hogy milyen lesz a mintában az egyes kategóriák eloszlása, míg a másik esetben, hiszen valószínűségeken alapul, ez nem garantálható (Olsen et al., 2012). Ezt a szempontot azért részesítettük előnyben, mert

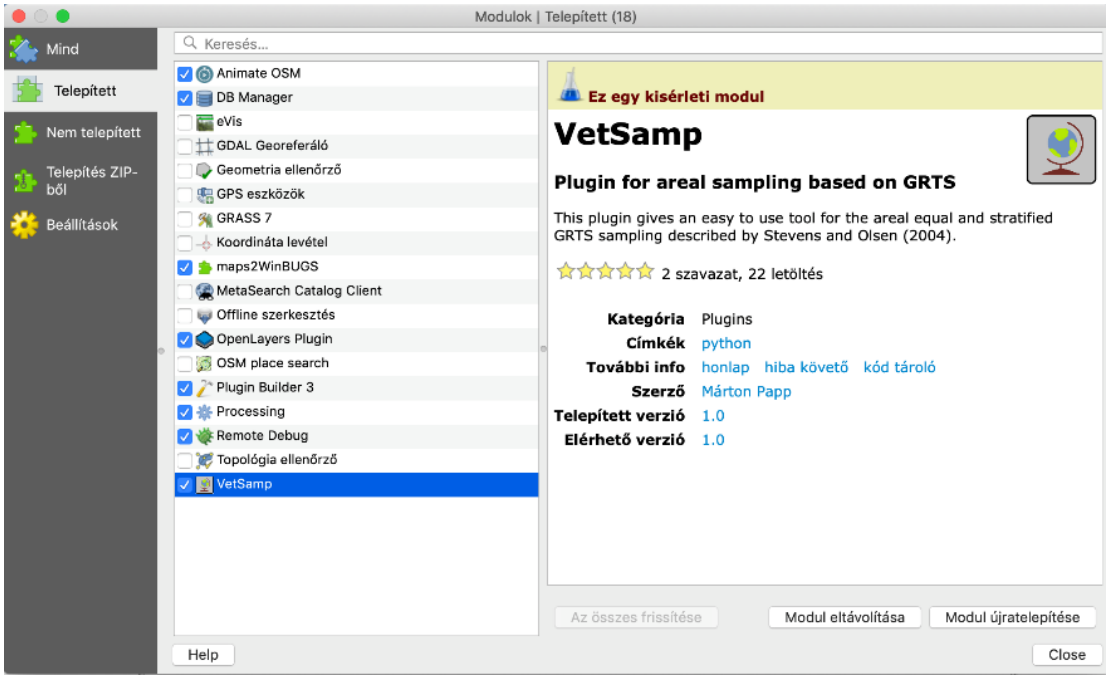
hazánkban a gyakran korlátozottan rendelkezésre álló anyagi források következtében általában csak kis számú minta vételére van lehetőség, ezért a kívánt eloszlás így könnyebben garantálható.

A vizsgálat szempontjából fontos populációs változók, a minta segítségével, a Cordy által folytonos populációkra kiterjesztett Horvitz-Thompson becslés segítségével határozhatók meg (Stevens és Olsen, 2004).

5 Az eredmények ismertetése

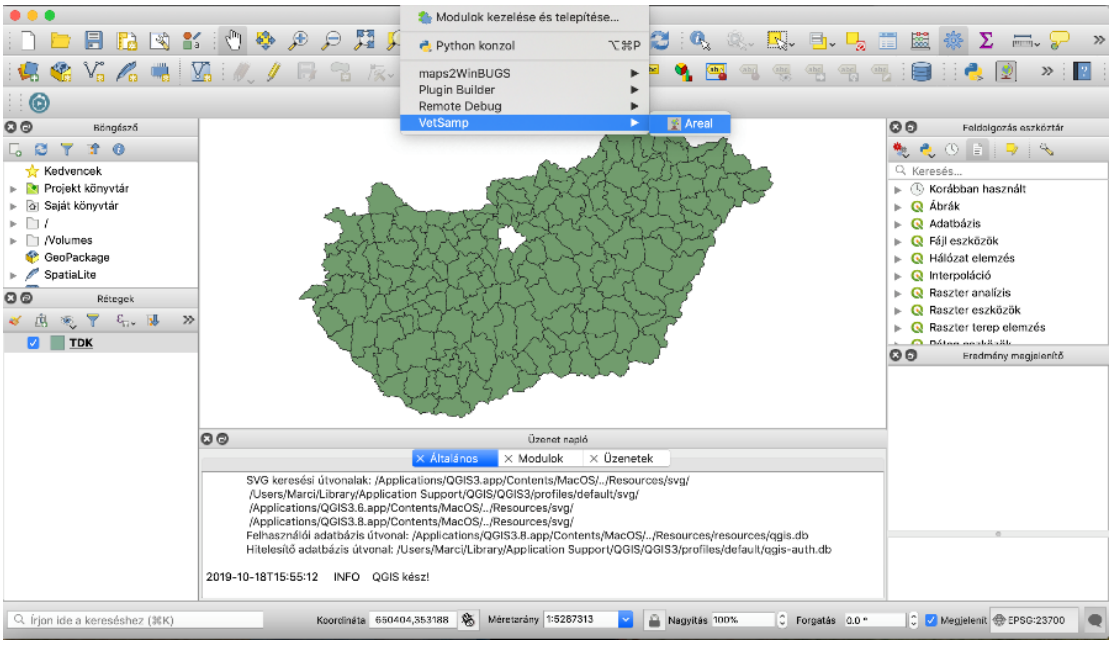
Eredményeinket, vagyis az általunk elkészített QGIS Python plugint egy esettanulmányon keresztül szeretném bemutatni. Tegyük fel, hogy egy vizsgálathoz kullancsokat szeretnénk gyűjteni, olyan módon, hogy az egész ország területére vonatkozóan következtetéseket vonhassunk le a kullancspopulációról. Ehhez a mintavételezési pontokat térben úgy kell kijelölni, hogy a mintából biztos becsléseket nyerhessünk a populáció tekintetében. Stevens és Olsen (2004) módszere, a GRTS kiválóan alkalmas erre. A munkánk során készített QGIS pluginnal, mely ennek az elvén alapul, egyszerűen elvégezhetjük a mintavételezési helyek kijelölését.

Amennyiben nincsen telepítve a plugin, ez könnyen megtehető a QGIS **modulok** menüjének **modulok kezelése és telepítése…** menüpontja segítségével. Amennyiben a kísérleti modulok engedélyezve vannak, a keresőben a plugin nevére rákeresve telepíthető (1. ábra).



1. ábra
A plugin letöltése a plugin könyvtárból

Ha a program telepítve van, akkor a **modulok** menüben megjelenik a **VetSamp** menüpont. A kurzort ide vezetve az **Areal** almenü kiválasztásával indítható a plugin (2. ábra).

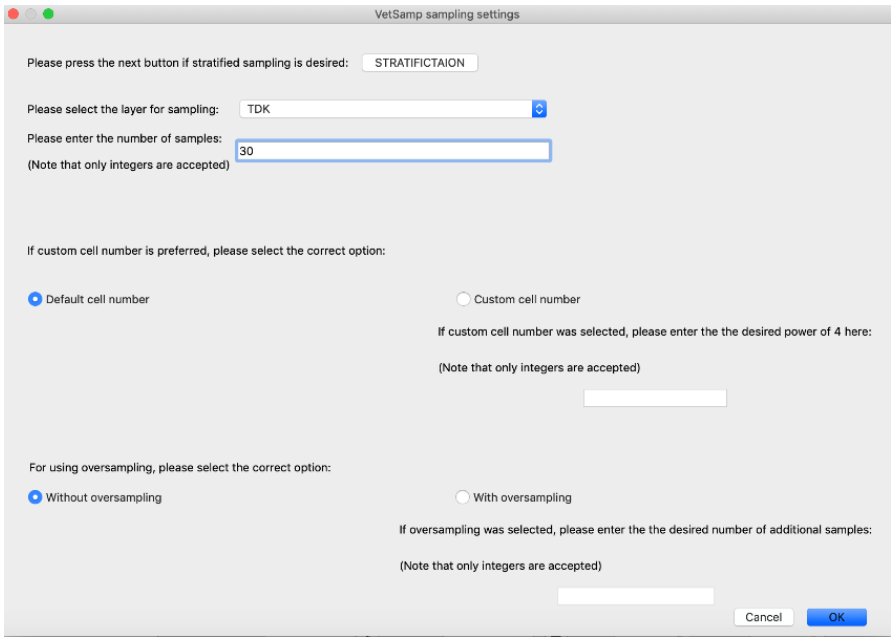


2. ábra
A plugin megnyitása

5.1 Egyenlő valószínűségű mintavételezés:

Az első példában úgy jelölünk ki mintavételezési helyeket, hogy azok a populáció egészére nézve egyenlő valószínűséggel kerüljenek kiválasztásra. A példa során Magyarország járástérképét használjuk, melyen Budapest ki van hagyva, hiszen a vizsgálat céljai szerint ez a terület kevés információtartalommal bír.

Amennyiben a térképi réteget betöltöttük, amely jelen esetben .shp, vagyis ESRI shapefile formátumú és a TDK nevet viseli, a következő lépés a plugin elindítása. Ezt követően megjelenik a fő párbeszédablak (3. ábra). A jelenlegi céloknak megfelelő mintavételhez szükséges beállítások ezen az ablakon megtehetők. Az első lépés ezek között a réteg megadása. Itt egy legördülő listában kell kiválasztani a használni kívánt réteg nevét. Jelen vizsgálatban 30 mintát szeretnénk venni, ennek a számát be kell írni az erre szolgáló mezőbe. A program csak nullánál nagyobb egész számot fogad el, egyéb esetben hibaüzenet jelenik meg. A beállítások után az **OK** gombra kattintva futtatható le az algoritmus.



3. ábra
Egyenlő valószínűségű mintavétel esetén
a mintaelemszám és a réteg megadása

A plugin ezután meghatározza a térképi réteg kiterjedését az X és az Y tengelyen is, ezek közül a nagyobbat választja a négyzetháló oldalhosszának. A minimálisan szükséges cellák száma 4^m, $m = \log_4(n)$, ahol n a kívánt mintaelemek száma. A plugin m-hez kettőt hozzáad a cellaszám kiszámítása során. Ennek oka, hogy a négyzetháló random felhelyezéséhez X vagy Y tengely felé (amelyik a kettő közül hosszabbnak bizonyult) egy cellával túl kell lógnia a réteg kiterjedésén és minél kisebb a cella nagysága, annál kevesebb használatlan terület keletkezik.

A fentiek szerint a négyzetháló kiterjedését X és Y tengely irányába meg kell hosszabbítani úgy, hogy az egy cellával nyúljon túl a réteg leghosszabb kiterjedésén. Ennek a mértéke $l\frac{2^m}{(2^m-1)}$, ahol l egy cella hossza, mielőtt a négyzethálót meghosszabbítanánk, 2^m pedig egy sorban található összes cella száma. A felhasználó a cellaszámot módosíthatja. Amennyiben **deafault cell number**-ről **custom cell number**-re kapcsol, megadhatja négynek azt a hatványkitevőjét, amely a cellaszámot meghatározza. A plugin hibaüzenetet ad, amennyiben ez a hatványkitevő kisebb, mint amekkorát a fenti összefüggés szerint automatikusan meghatároz.

A meghosszabbított négyzetháló random módon kerül a rétegre. Ezután az egyes cellákat a plugin kvadráns rekurzív módszerrel elnevezi. Az egyenes hosszúsága, melyen a celláknak és a polygonoknak megfelelő szakaszok ki lesznek jelölve, *100000x* a kívánt mintaelemszám. A GRTS algoritmusban az egyenes a mintaszámmal egyenlő hosszúságú

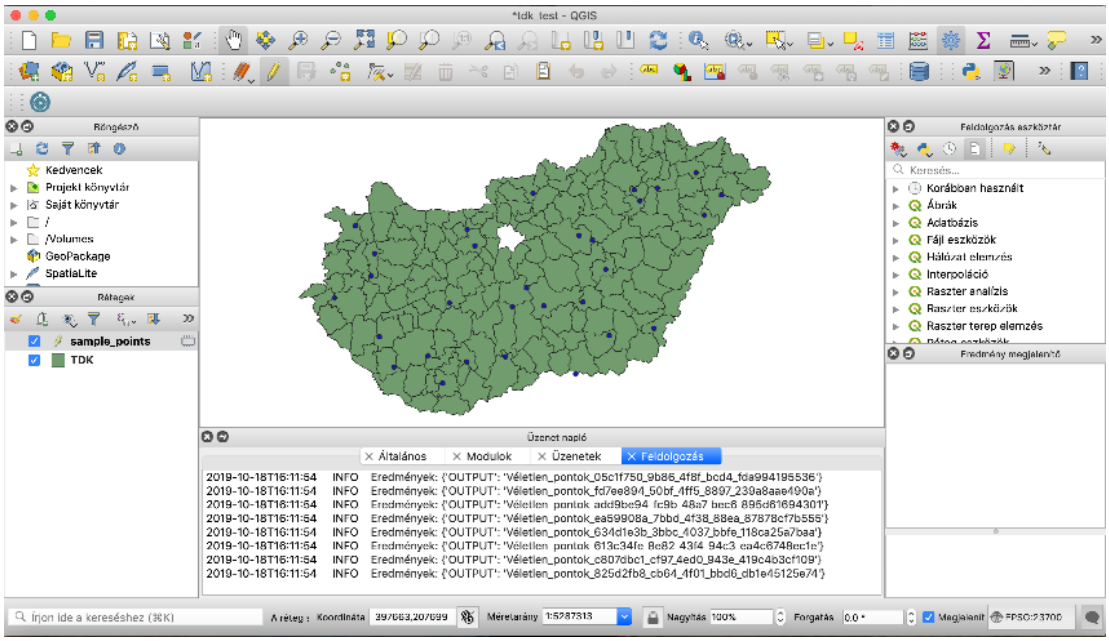
és két tizedes pontosságig történik a pontok kijelölése. Az általunk készített alkalmazásban egész számokat jelölünk ki. Miután a teljes egyenes *100000*-szerese a mintaszámnak, ezért három nagyságrenddel nagyobb pontosságot kapunk, mint az eredeti módszer esetén (Olsen et al., 2012).

A cellák egyenesen elfoglalt helyét azok sorszáma határozza meg, melyet a kvadráns rekurzív módszerrel történt sorba rendezés során nyertünk.

A cellához tartozó szakasz hosszúsága a benne foglalt polygon részletek egyenesen elfoglalt hosszainak összegével egyenlő. A polygon részletekhez tartozó szakasz akkora, amekkora a polygon teljes egyenesen elfoglalt területének olyan arányú része, mint a polyon részlet és a teljes polygon aránya. Egy polygon egyenesen elfoglalt teljes hossza akkora része az egyenesnek, amekkora a polygon területének aránya a teljes réteg területéhez. A cellához tartozó szakaszon az egyes polygon részletek sorrendje random.

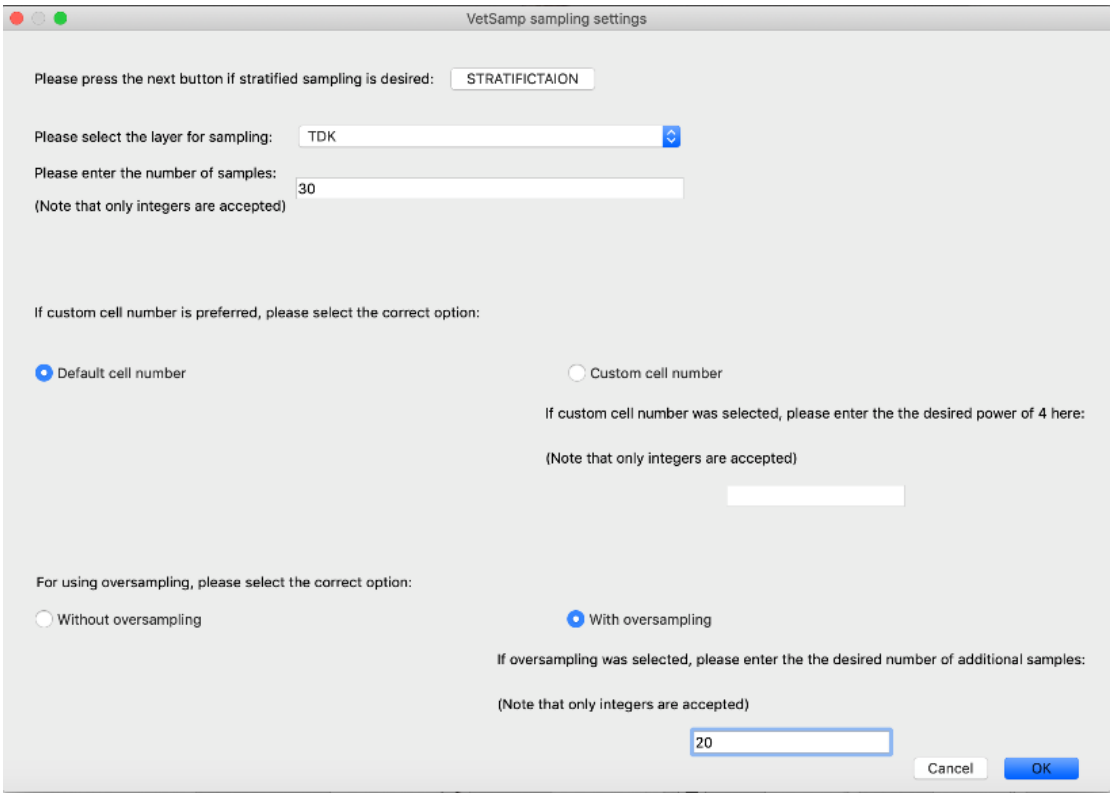
A mintavételezés során az egyenesen az első mintapont véletlenszerűen kerül kiválasztásra. Ezt követően minden *k* pontban egy következő elemet is kiválaszt a plugin, $k = \frac{N}{n}$, ahol *n* a mintaelemszám és *N* az egyenes teljes hossza. A pont amelyik cellához és azon belül amelyik polygon részlethez tartozó szakaszra esik az egyenesen, a mintaelem ebben a részletben kijelölt random pont lesz.

A mintaelemek egy új rétegen belül kerülnek rögzítésre, **sample_points** néven. Itt az attributum tábla négy információt tartalmaz, a cella számát, amibe a pont került, a pont sorszámát amintákat tartalmazó rétegben és a réteg vetületének megfelelő X és Y koordinátákat. A fentiek szerint lefuttatva a mintavételezést a 4. ábrán látható eredményt kaptuk.



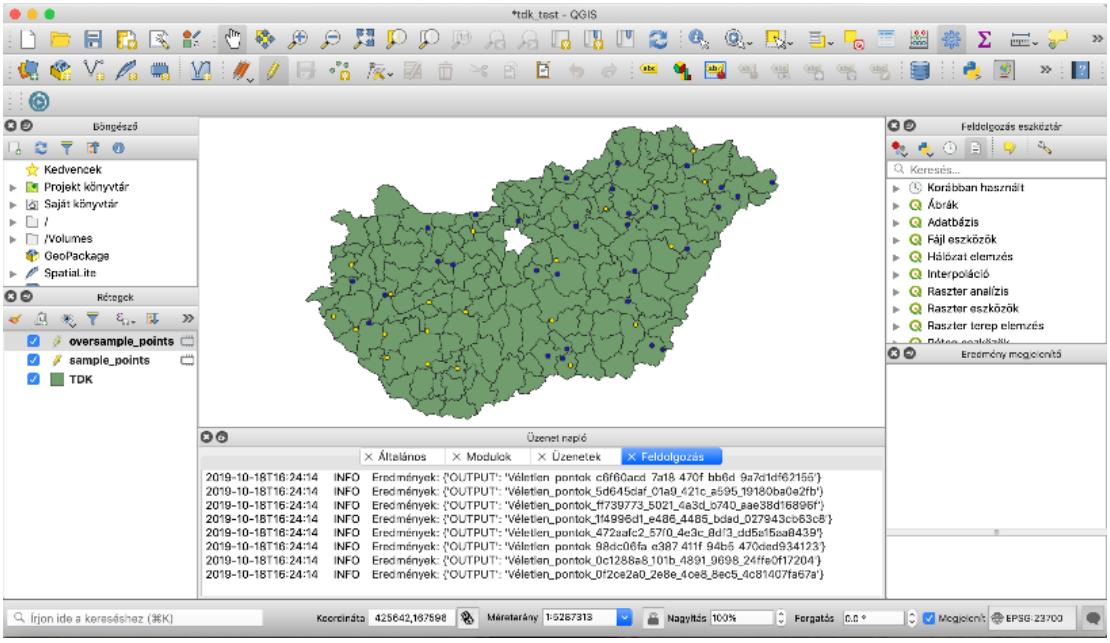
4. ábra
A kijelölt mintapontok (kék színnel)
a térképi rétegen

Elképzelhető, hogy egyes pontok a mintavétel megvalósításakor kiesnek, mert nem sikerült kullancsot gyűjteni onnan, vagy már a megközelítés során fizikai, illetve egyéb akadályokba ütköztünk. Ezekre a lehetőségekre a kísérlettervezés során fel kell készülni, hiszen a minta későbbi kiegészítése már lehetetlen, vagy csupán tudatos módosítással lehetséges, mely torzíthatja a becsléseinket. A módszer és a plugin is alkalmas tartalék minták vételére. Ahhoz, hogy a plugin ilyen módon végezze el a mintapontok kijelölését, át kell kapcsolni **without oversampling**-ből **with oversampling**-re, majd a megfelelő mezőben meg kell adni a tartalékminták kívánt számát, mely legyen ez esetünkben 20 (5. ábra).



5. ábra
Mintavételezés tartalék pontok kijelölésével

A plugin ebben az esetben úgy végzi el a fent leírt mintavételezést, hogy a mintaszámhoz hozzáadja a tartalék minták számát, majd eszerint fut le az algoritmus. A fent ismertetetthez képest azonban a cellák sorszámozása után a sorszámok egyes helyiértékein található számokat megfordítja, vagyis a kisebb helyiértéken lévő kerül a nagyobb helyére és fordítva. A cellákhoz tartozó szakaszok az egyenesen ezen új sorszámok szerint kerülnek rendezésre. A mintavétel végén csak annyi pont kerül a **sample_ponints** rétegbe, amennyi mintaszám meg van adva, a maradékot az **oversample_points** nevű rétegbe helyezi a plugin (6. ábra). Ebben a rétegben is ugyanazok az információk vannak rögzítve a pontokról, mint a mintapontokat tartalmazóban. Amennyiben a mintavételezés során egyes pontok kiesnek, ebből a rétegből újak emelhetők a helyükre.



6. ábra
Tartalék mintákat is igénybe vevő mintavételezés
után a mintapontok (kék színnel) és a tartalék pontok (sárga színnel) a rétegen

5.2 Egyenlőtlen valószínűségű mintavételezés:

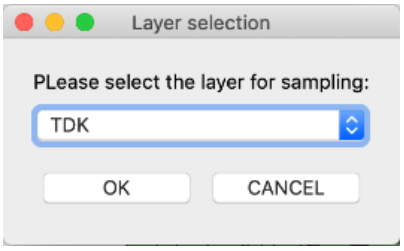
A vizsgálat során szeretnénk a csapadék és az átlaghőmérséklet hatását is figyelembe venni. Ehhez szükséges, hogy minden kategóriából elegendő mintaelem álljon rendelkezésre. A polygonokat négy csoportba szeretnénk rendezni, az évi összes csapadék 10 éves átlaga és a 10 °C fölötti hőmérsékletek éves átlagának 10 éves átlaga alapján. Mindkét kategóriában egy magas és egy alacsony csoportot hozunk létre, aszerint, hogy a medián fölött vagy alatt van az érték. Az így létrehozott 4 kategóriában egyaránt 10 mintát és 5 tartalék mintát szeretnénk kijelölni.

A plugin rétegzett mintavételi módjával lehetőség van a rétegben található polygonokat bizonyos rájuk jellemző tulajdonságok alapján csoportosítani. Folytonos és nem folytonos változók alapján is lehetséges kategóriákat létrehozni. A plugin először ellenőrzi, hogy a változó csak számokat tartalmaz-e vagy mást is. Amennyiben csak számokat tartalmaz, akkor a felhasználónak kell kiválasztani, hogy folytonos vagy nem folytonos változóként szeretné felhasználni azt a csoportosítás során. Folytonos változók esetében a felhasználónak meg kell adnia az intervallumot és a hozzá tartozó kódot, amely a kategóriát jelöli. Az intervallumnál megadott felső érték a következő intervallum kezdete, kivéve az utolsót. Nem folytonos változók esetén az összes lehetséges értékhez, amit a változó felvehet, kódot kell rendelnie a felhasználónak.

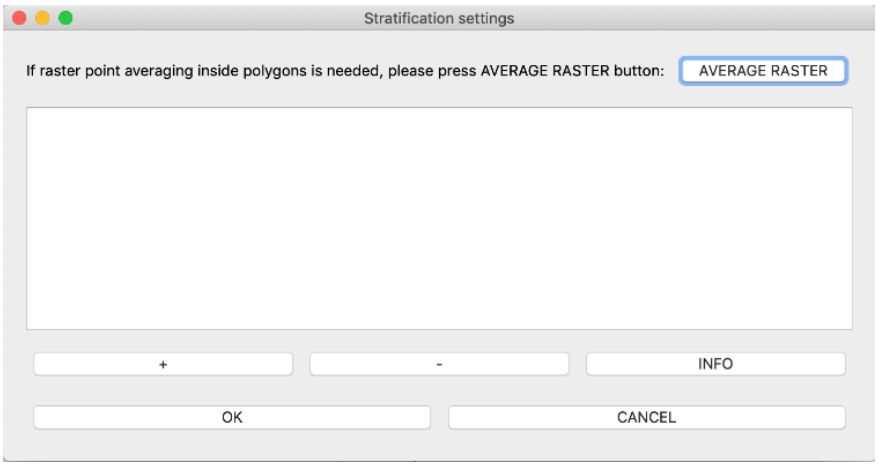
Az adott polygon végső kategóriája az egyes kategorizáló változók szerint a polygonhoz tartozó kódok egymás után illesztésével jön létre. Következő lépésben a felhasználónak ki kell választania, hogy szeretne-e tartalék pontokat, majd meg kell adnia az egyes kategóriákon belül kívánt minta- és tartalékpont mennyiséget. A plugin ezután elvégzi a mintavételezést a kategóriákon belül. A kijelölt mintaelemeket tároló rétegek neve **sample_points_** és a kategória neve, míg a tartalék pontok esetében **oversample_points_** és a kategória neve.

A vizsgálat során első lépésben meghatározzuk a mediánokat, melyek az egyes változók során a magas és alacsony kategóriák határát adják meg. Az attribútum táblában **gdd**-vel jelölt átlagos 10 ºC feletti hőmérséklet mediánja 11.78345, míg az **y10prec**-el jelölt átlagos csapadéké 591.60135, ezek az értékek választják tehát el a magas és alacsony kategóriákat. A magas hőmérsékleti kategória MH-, az alacsony pedig AH-, míg a magas csapadék kategória MCS-, az alacsony ACS- kódot kap. Ezáltal négy végső kategória jön létre, az AHACS, AHMCS, MHACS és MHMCS.

A plugin elindítása után a fő párbeszédablakon a **STRATIFICATION** gombra kell kattintani a rétegzett mintavételhez. Ezután a megjelenő ablakban ki kell választani a réteget (7. ábra), amelyen a mintavételt el szeretnénk végezni, majd ezt követően a kategóriák regisztrálására szolgáló ablak jelenik meg (8.ábra).

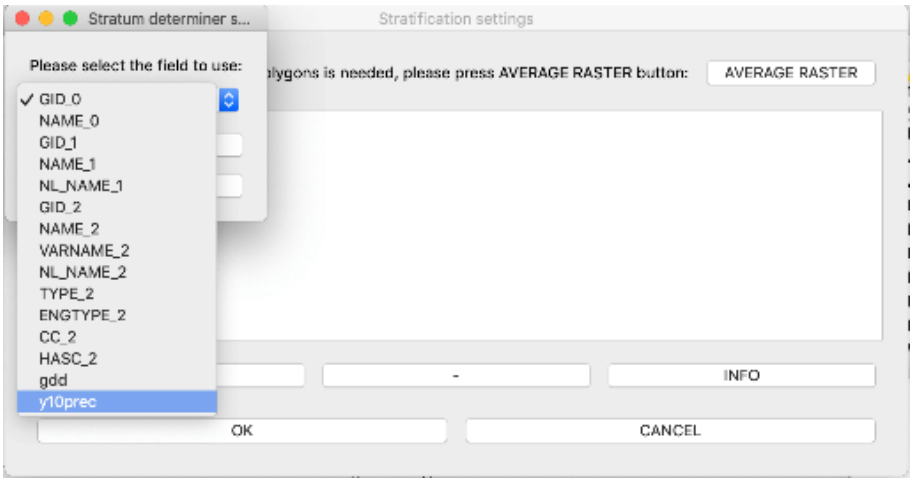


7. ábra
A réteg kiválasztására szolgáló párbeszédablak



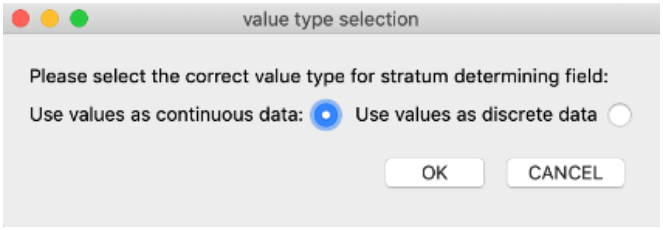
8. ábra
Rétegek regisztrálására szolgáló párbeszédablak
a + gomb segítségébel új kategória adható meg,
a – gombbal törölhető a kiválasztott kategorizáló változó,
míg az INFO gombbal további információ jeleníthető meg róla

A + gomb segítségével adható meg új kategória. A mező megadása (9. ábra) után a program ellenőrzi, hogy csak számot tartalmaznak-e a mezőben található értékek, vagy mást is. Amennyiben nem csak számot tartalmaznak, nem folytonos változóként kezeli tovább a program. Jelen esetben a kiválasztott mezőkben található értékek mindkét esetben számok.



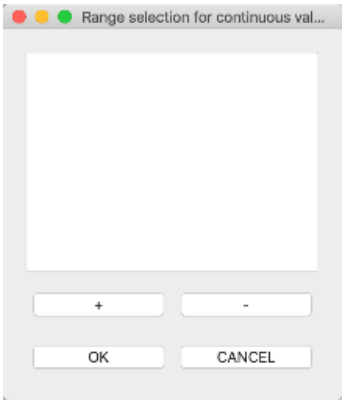
9. ábra
Annak a mezőnek a kiválasztása, amely alapján
a kategóriát létre szeretnénk hozni

A következő ablakban meg kell jelölni, hogy folytonos vagy nem folytonos változóként kezelje-e a plugin a mezőt. Mindkét változót folytonos formában szeretnénk használni, ezért a **use values as continuous data**-ra kell kapcsolni (10. ábra).

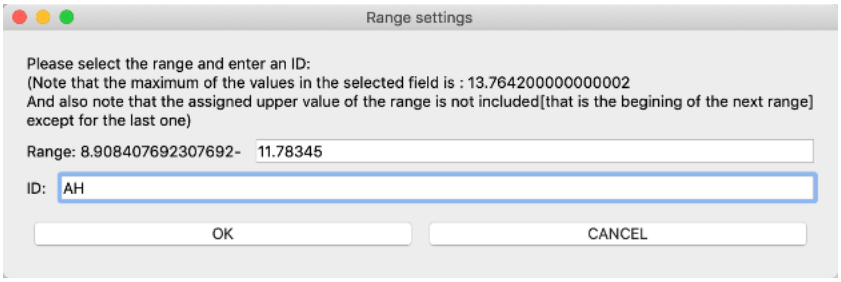


10. ábra
Párbeszédablak, mely segítségével kiválasztható, hogy a változót folytonos vagy nem-folytonos formában szeretnénk-e használni

A megjelenő ablakban (11. ábra) adhatóak meg az intervallumok és azok kódja, a + gomb segítségével (12. ábra). Intervallum törlésénél az összes kijelöltnél nagyobbat is törli a plugin, hiszen azok köztes hiányában értelmezhetetlenek. Törölni a kívánt intervallumot, annak kiválasztása után a – gomb segítségével lehet. Az utolsó intervallum megadása után az **OK** gombbal lehet regisztrálni a kategóriát.



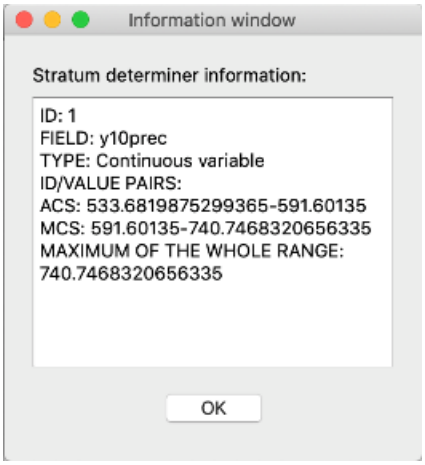
11. ábra
A kategóriát meghatározó intervallumok megadására szolgáló ablak
A + gomb segítségével adható meg új intervallum,
a – gombbal pedig törölhető a kiválasztott



12. ábra
Az intervallum megadásakor annak felső értékét kell megadni,
azonban úgy, hogy ez a beírt érték a következő első eleme lesz,
(kivéve az utolsó intervallumot)
az intervallum definiálása mellett a kategória nevét is meg kell adni

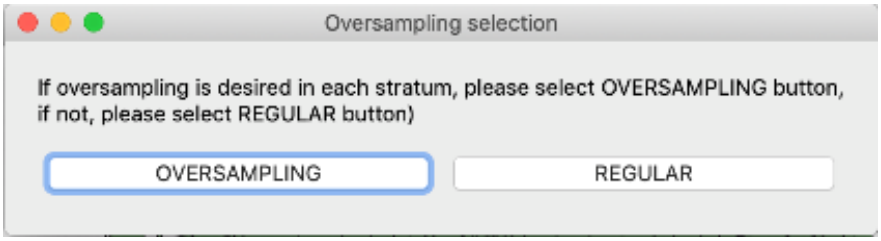
A kategóriák az azokat tároló ablakon törölhetők a – gombbal, illetve további információ jeleníthető meg róluk az **INFO** gomb segítségével, miután a megfelelő sort kijelöltük (13. ábra)

Amennyiben szükséges, az **AVERAGE RASTER** gomb segítségével átlagolhatók egy raszter réteg értékei a polygonokon belül.



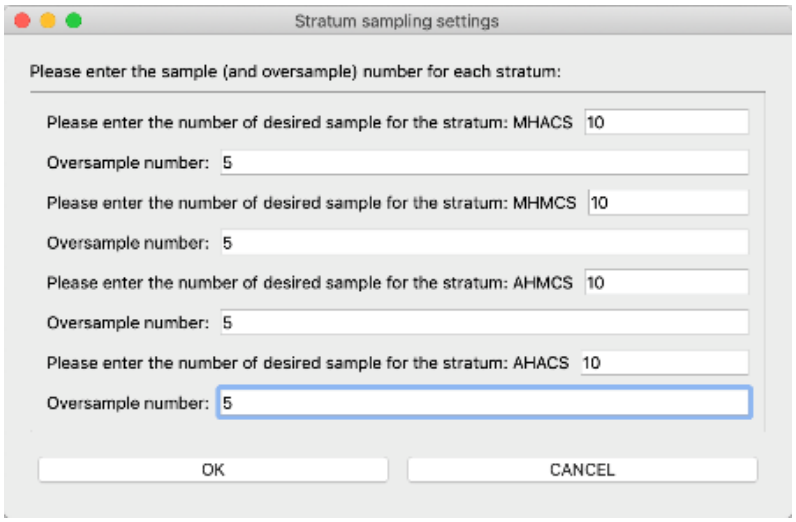
13. ábra
Az INFO gomb megnyomása után több információ megjeleníthető a kijelölt kategóriáról

A csoportok elkészítése után az **OK** gombra kattintva lehet továbbmenni. A következő ablakban kell kiválasztani, hogy tartalék mintákat is szeretnénk-e a mintavételezés során (14. ábra).

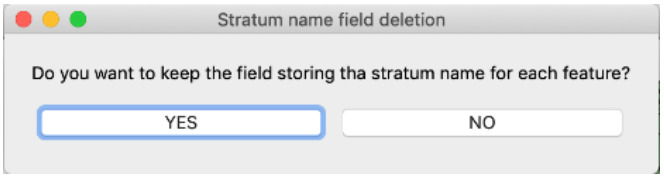


14. ábra
Párbeszédablak, melyen ki kell választani, hogy tartalékpontokkal, vagy azok nélkül szeretnénk-e elvégezni a mintavételezést

Jelen esetben tartalék mintákat is szeretnénk, ezért az **OVERSAMPLING** gombra kell kattintani. Miután megadtuk az egyes kategóriákban kívánt minta- és tartalékminta mennyiséget, mely jelen esetben 10 rendes és 5 tartalék minta minden kategóriában (15. ábra), az OK gombra kattintva futtatható le a mintavételezés. A plugin utána megkérdezi, hogy megtartsa-e azt a mezőt, amiben az egyes polygonokhoz tartozó kategóriát tárolja (16. ábra). A kísérlet dokumentálása szempontjából előnyösebb ezt megtartani, ezért jelen esetben is ezt választjuk.

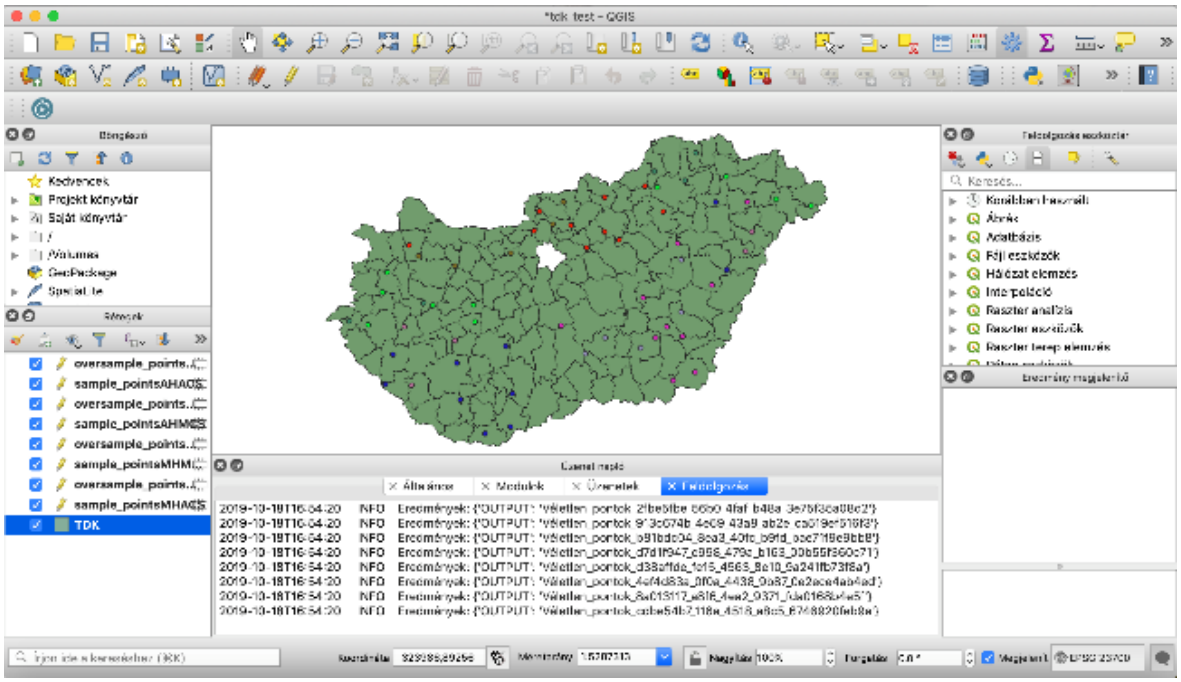


15. ábra
Meg kell adni minden réteghez az azon belül kívánt mintapont- és tartalékminta számát



16. ábra
Párbeszédablak, melyen ki kell választani, hogy a kategóriákat tartalmazó mezőt kitörölje-e a plugin

Az elkészült rétegek a plugin futása után megjelennek (17. ábra), a kategóriákat tartalmazó mező pedig az eredeti réteg attribútum táblájában megtalálható **str_nm1** néven (18. ábra).



17. ábra
Az elkészült mintapontok és tartalék minták
kategóriák szerint eltérő színnel a rétegen

gdd	y10prec	str_nm1
12.0904846...	580.702089...	MHACS
12.4057538...	568.236690...	MHACS
12.4320846...	583.090556...	MHACS
12.5918000...	568.916965...	MHACS
12.5073000...	563.413769...	MHACS
12.61773076...	570.309419...	MHACS
12.22107692...	573.1357797...	MHACS
12.5324230...	564.393825...	MHACS
11.52217692...	565.072441...	AHACS
11.69141538...	597.720843...	AHMCS
11.9069384...	599.208986...	MHMCS
11.5850384...	605.157940...	AHMCS
11.4990769...	577.003264...	AHACS
11.57533076...	533.681987...	AHACS
10.81746153...	588.613728...	AHACS
11.57526923...	552.919082...	AHACS
11.61732307...	586.121625...	AHACS
11.91930769...	579.489468...	MHACS
11.69141538...	597.720843...	AHMCS
13.22256153...	568.557013...	MHACS
11.7696692...	587.4103702...	AHACS

18. ábra
Részlet a TDK réteghez tartozó attribútum táblából,
melyen a hőmérékletre és csapadékra vonatkozó adatok találhatók,
valamint a kategóriák az első néhány polygon esetében

6 Megbeszélés

A Stevens és Olsen (2004) által ismertetett GRTS algoritmus kiválóan alkalmas földrajzi minták kijelölésére, ezért hatékonyan alkalmazható a legváltozatosabb tudományterületeken. Ennek okán széles körben elterjedt és alkalmazott mintavételezési módszer, akár vízi, légi vagy szárazföldi mintáról van szó (Brown et al., 2015). Jól használható az állatorvos-tudományokban is, hiszen sok esetben itt is igénybe vesznek földrajzi mintát, például epidemiológiai-, parazitológiai- vizsgálatok során. Hasznos a biológiai kutatások számára is, mert ezek a területek még gyakrabban próbálnak térben kiterjedt populációkról következtetéseket levonni. Nem vitatható a módszer szerepe a természettudományok más területein sem, ahogyan felhasználható akár a szociológia és sok más tudományos kutatás tervezése során. Ezen okonál fogva ki kell emelni a jelentőségét a módszer minden olyan alkalmazásának, mely a gyakorlat számára elérhetővé teszi.

A GRTS felhasználása eddig egy R-es programcsomag (’spsurvey’ csomag) segítségével volt megvalósítható, mely több gyakorlati nehézséget is magában rejt. Azon kívül, hogy az algoritmus ezen alkalmazása mélyebb R-es felhasználói ismereteket igényel, ezzel korlátozva azok körét, akik igénybe tudják venni, az R környezet nem tesz lehetővé olyan rugalmas térképkezelést, mint amelyet egy GIS szoftver nyújthat. Ennek következtében gyakran több program együttes alkalmazását igényli a probléma megoldása.

Munkánk során szerettünk volna a fenti nehézségekre megoldást kínálni olyan módon, hogy a felhasználó csupán egyetlen szoftver felhasználásával el tudja végezni a teljes mintavételi folyamatot, miközben a GIS rendszerek nyújtotta előnyöket is élvezni tudja, valamint az algoritmus paraméterezése sem igényel a szükségesnél nagyobb felkészültséget.

Azért esett választásunk a QGIS programra a GIS szoftverek közül, mert ez ingyenesen hozzáférhető mindenki számára, hatékony térképkezelést tesz lehetővé, ezért a mintavételezéshez használt réteg különösebb nehézségek nélkül előállítható és módosítható, valamint a felhasználók közössége pluginokon keresztül folyamatosan fejleszti az elérhető funkciókat. Ezeknek köszönhetően a vizsgálat előkészítése nagy valószínűséggel egyetlen program használatával elvégezhető. A GRTS módszer általunk készített alkalmazását mi is egy QGIS Python plugin formájában hoztuk létre.

A plugin fejlesztése során arra törkedtünk, hogy a módszer felhasználói szintű ismeretével mindenki gond nélkül testreszabhassa a mintavételezés beállításait. A grafikus felhasználói felületeket megpróbáltuk úgy felépíteni, hogy azok a mintavétel folyamatának logikáját kövessék és könnyen értelmezhetők legyenek a felhasználó számára. Amennyiben a beállítások során olyan adatok kerültek megadásra, amelyek formátuma nem megfelelő az adott helyzetben a program számára, vagy más módon hibát generálnak, akkor a hiba típusáról a plugin egy ablakban tájékoztatja a felhasználót, majd visszavezeti ahhoz a ponthoz, ahol a hiba keletkezett. Ennek segítségével, reményeink szerint a kívánt beállítások gördülékenyen elvégezhetők.

Az általunk készített plugin jelentőségét a fentiek mellett kiemeli, hogy jelen pillanatban a módszer egyetlen GIS programon belül létrehozott alkalmazása. Szeretnénk a plugin fejlesztéseivel és az esetlegesen felmerülő hibák folyamatos javításaival egy olyan eszközt biztosítani a kutatók számára, amely megbízható és hasznos a kutatások tervezése során.

Reményeink szerint a plugin jól alkalmazható lesz sok állatorvosi, orvosi, biológiai és egyéb vizsgálat tervezése során. Reméljük továbbá, hogy a segítségével a szakmai szempontoknak a lehető legkevésbé szabnak gátat a gyakorlati nehézségek. Szeretnénk, ha a plugint felhasználva a kísérlettervezés során az egyes szakterületek képviselői külső segítség igénybe vétele nélkül el tudnák végezni a mintavételi helyek kijelölését, mint a vizsgálat tervezésének egy igen jelentős részét, ezáltal leegyszerűsítve a tudományos munka komplex folyamatát. További célunk és reményünk, hogy munkánk segítségével a kutatások folyamán olyan minták és adatok képzik majd a statisztikai elemzések alapját, amelyek reprezentatívak a vizsgált populációra, ezzel növelve azok minőségét.

Az általunk készített plugin jelen pillantaban csupán területeken tud mintavételi pontokat kijelölni egyenlő valószínűségű mintavétel és rétegzett mintavétel során. Szeretnénk a jövőben a funkciókat kibővíteni lineáris és pont objektumokról történő mintavételezésre is.

A QGIS plugin könyvtárában elérhető az általunk készített plugin hibakövetője. Ezen keresztül a felhasználóknak lehetősége van tájékoztatni minket a felmerülő hibákról illetve olyan új funkciók bevezetését is javasolhatják, amelyek a gyakrolati felhasználás során, mint hasznos változtatások felmerültek. A visszajelzések alapján lehetőségünk van a plugint folyamatosan frissíteni, hogy a gyakorlati felhasználói igényeket a lehető legjobban kielégíthessük. A legfrissebb változtatásokat először a plugin kód tárolójába töltjük fel (az ehhez vezető link is elérhető a plugin könyvtárban), innen ezek a verziók

akár telepíthetők is. A plugin gerincét képző kódrészlet legfrissebb verziója megtekinthető a mellékletben. A fejlesztés során elkészült stabil verzókat pedig feltöltjük a QGIS Python plugin könyvtárba, ahol ezek a fentebb ismertett módon telepíthetők és felhasználhatók.

A földrajzi minták helyes kijelölése az ezeken alapuló kutatások alapvető része, hiszen segítségével minimalizálható a térbeli kapcsolatokból eredő bizonytalanság a vizsgálat során nyert adatokban. Reményeink szerint az erre nagyon jól alkalmazható GRTS módszer általunk készített alkalmazása kielégíti a felhasználók igényeit és hasznos eszköze lesz a legváltozatosabb tudományos kutatásoknak.

7 Összefoglalás

Új ismeretek szerzése a legtöbb tudományterület számára elképzelhetetlen lenne minták felhasználása nélkül. Azonban a tudományon kívül a gazdaság, a politika és a mindennapi életünk számos más része is mintákat alapul véve hoz döntéseket. Kijelenthetjük, hogy a mintából való extrapoláció egyidős az emberiséggel, annak mindennapos tevékenységéhez tartozik, csupán a minta minőségében, használhatóságában, elbírálásának mikéntjében figyelhetők meg jelentős különbségek.

A jó mintától általában az várható el, hogy segítségével pontos képet kaphassunk a populációról, amelyből származik. Leggyakrabban ezt nevezik reprezentatív mintának, mely lényegében a populáció miniatűrjének tekinthető.

Ahhoz, hogy a minta megfeleljen ennek a kritériumnak, olyan módszerrel kell hozzájutni, amely megőrzi az eredeti eloszlást. Ezek a módszerek általában random választanak a populáció elemei közül. Ilyen például az egyszerű random mintavétel, illetve a szisztematikus random mintavétel. Ezek reprezentatív technikák, de nem mindig eredményeznek reprezentatív mintát. Előfordulhat, hogy bizonyos elemei a populációnak felülreprezentáltak lesznek, illetve, ha kétdimenziós térben dolgozunk, aggregált mintát eredményezhetnek.

Megállapítható tehát, hogy térben végzett mintavételezés során a mintaelemek térbeli kapcsolata nem elhanyagolható. Az a minta, amely egyenletes, sokkal reprezentatívabbnak tekinthető, hiszen ezzel az autokorelláció minimalizálható.

Időnként azonban célszerű az eredeti eloszlástól eltérő mintát venni: például amikor egyes elemek aránya alacsony, azonban jelenlétük fontos a vizsgálat során. Különösen igaz ez olyan helyzetekre, amikor korlátozott számú minta vételére van lehetőség.

A mintavétel folyamatát különféle gyakorlati nehézségek is akadályozhatják, elsősorban földrajzi mintavételezés során. Az előre meghatározott mintavételi helyek a legnagyobb óvatosság ellenére is könnyen eshetnek magánterületre, vagy fizikailag megközelíthetetlen részekre. Olyan modellt érdemes használni, amely alkalmas ezeknek a váratlan helyzeteknek a kezelésére is.

A fenti tulajdonságok jellemzik azt a módszert, melyet Stevens és Olsen 2004-ben publikált és GRTS-nek, azaz Generalised Random-Tesselation Stratified modellnek nevezett el. Munkánk során a fenti módszer azon részének alkalmazását készítettük el, mely kétdimenziós térben generál mintaelemeket. Lehetőség van egyenlő

valószínűséggel vett minta és rétegelt mintavételezés használatára is. Munkánkat a QGIS ingyenes térinformatikai szoftverhez tartozó plugin-ként készítettük el. Célunk az volt, hogy a program egyszerűen kezelhető grafikus felhasználói felületen legyen vezérelhető, és mindenki számára elérhető, így könnyen hasznosítható legyen elsősorban az állatorvosi epidemiológia, parazitológia, ökológia területén. Többek között ezért is esett a választásunk a QGIS használatára. A plugin jelenlegi verziója elérhető a QGIS plugin könyvtárában VetSamp néven.

8 Summary

Gathering new informations in most scientific fields would be impossible without the use of samples. However not only science depends on sampling but economy, politics and everyday life make decisions daily based on samples. We can conclude that extrapolation from samples originate from early humanity although there are explicit differences in the quality, usability and interpretation of samples.

A sample is considered good when we can make proper assumptions about the population from it originates, based on it. Mostly it is called a representative sample, which is basically the miniature of the population.

To make sure that the sample has the above characteristics we need methods for obtaining it that keeps the original distribution among the elements. These are mostly take elements in a random way. The most well-knowns are the simple random sampling and the systematic random sampling. Despite they are representative techniques they not always result in representative samples. Sometimes different types of elements are overrepresented or aggregated points can be observed in the two-dimensional space.

We can state that the connection between sample elements is not negligible when sampling in two-dimensions. A sample that is well distributed over the sampling area is more likely representative as it is minimizing autocorrelation.

Sometimes however it is favourable to use different distribution: for example when some types of the elements take a small ratio of the whole population but are important for the study. It is especially important in cases, when limited sample numbers are accessible.

Technical difficulties can also occur especially when geographical sampling is presented. Sampling locations can easily be placed in private property or can be physically inaccessible. It is helpful to use methods that can deal with these situations.

These features are representing the method described by Stevens and Olsen in 2004. They named it Generalised Random-Tesselation Stratified (GRTS) model. We made an application of the areal equal and stratified sampling part of this method. We have constructed it as a plugin for QGIS, a free and open source geographical information system. Our goal was to make an easy to use graphical user interface for the program and to make it accessible and useful for everyone, especially in the field of veterinary epidemiology, parasitology and ecology. These are the main reasons we have chosen

QGIS. The current version of the plugin can be downloaded at the official QGIS plugin repository in the name VetSamp.

9 Köszönetnyilvánítás

Elsősorban szeretném megköszönni témavezetőmnek, Dr. Solymosi Norbertnek az útmutatását és segítségét a plugin és a dolgozat elkészítése során. Külön szeretném megköszönni páromnak, Csillának, hogy elviselt, amikor a kód hibái kötötték le minden figyelmemet, illetve köszönöm a dolgozat szerkesztésében nyújtott segítségét is. Szeretném megköszönni szüleimnek és családomnak a támogatást és a biztatást.

A projekt az Európai Unió támogatásával, az Európai Szociális Alap (ESZA) társfinanszírozásával valósul meg (a támogatási szerződés száma: AZ EFOP-3.6.3-VEKOP-16-2017-00005, címe: Tudományos utánpótlás erősítése a hallgatók tudományos műhelyeinek és programjainak támogatásával, a mentorálás folyamatának kidolgozásával).

10 Irodalomjegyzék

BROWN, J. A., ROBERTSON, B. L., MCDONALD, T., 2015: Spatially Balanced Sampling: application to environmental surveys. *Procedia Environmental Sciences*. 27. pp. 6-9.

COCHRAN, W. G., 1977: Sampling Techniques. 3rd ed. New York, John Wiley and Sons.

CORDY, C. B., 1993: An extension of the Horvitz-Thompson theorem to point sampling from a continuous universe. *Statistics & Probability Letters*. 18.5. pp. 353-362.

ETIKAN, I., MUSA, S. A., ALKASSIM, R. S., 2016: Comparison of Convenience Sampling and Purposive Sampling. *American Journal of Theoretical and Applied Statistics*. 5.1. pp. 1-4.

FULLER, W. A., 2009: Sampling Statistics. Hoboken, New Jersey, John Wiley and Sons.

GRAFSTÖRM, A., SCHELIN, L., 2013: How to Select Representative Samples. *Scandinavian Journal of Statistics*. 41.2. pp. 277-290.

GRIFFITH, D. A., 2005: Effective Geographic Sample Size in the Presence of Spatial Autocorrelation. *Annals of the Association of American Geographers*. 95.4. pp. 740-760.

HORVITZ, D. G., THOMPSON, D. J., 1952: A Generalization of Sampling Without Replacement From a finite Universe. *Journal of the American Statistical Association*. 47.260. pp. 663-685.

KRUSKAL, W., MOSTELLER F., 1979: Representative Sampling, III: the Current Statistical Literature. *International Statistical Review*. 47.3. pp. 245-265.

KRUSKAL, W., MOSTELLER, F., 1980: Representative sampling, IV: the History of the Concept in Statistics, 1895-1939. *International Statistical Review*. 48.2. pp.169-195.

MARK, D. M., 1990: Neighbor-based Properties of Some Orderings of Two-dimensional Space. *Geographical Analysis*. 22.2. pp. 145-157.

OLSEN, A. R., KINCAID, T. M., PAYTON, Q., 2012: Spatially balanced survey designs for natural resources. In: Gitzen, R. A., Millspaugh, J. J., Cooper, A. B., Licht, D. S. (eds.): *Design and Analysis of Long-Term Ecological Monitoring Studies*. New York, Cambridge University Press. pp. 126-150.

QGIS DEVELOPMENT TEAM, 2019: QGIS Geographic Information System. *Open Source Geospatial Foundation Project*. http://qgis.osgeo.org

SAMPATH, S., 2001: Sampling Theory and Methods. CRC Press.

SOLYMOSI, N., MEDVECZKY, I., 2000: Térinformatikai rendszerek alkalmazása járványtani vizsgálatokban és a fertőző betegségek kontrolljában. *Magyar Állatorvosok Lapja*. 122.8. pp. 504-507.

SOLYMOSI, N., REICZIGEL, J., BERKE, O., HARNOS, A., SZIGETI, S., FODOR, L., SZIGETI, G., BÓDIS, K., 2004: Spatial risk assessment of herd sero-status of Aujeszky's desease in a county in Hungary. *Preventive Veterinary Medicine*. 65.1-2. pp. 9-16.

SOLYMOSI, N., WAGNER, S. E., MARÓTI-AGÓTS, Á., ALLEPUZ, A., 2010: maps2WinBUGS: a QGIS plugin to facilitate data processing for Bayesian spatial modeling. Ecography. 33.6. pp. 1093-1096.

STEHMAN, S. V., OVERTON, W. S., 1994: Environmental Sampling and Monitoring. *Handbook of statistics*. 12. pp. 263-306.

STEPHAN, F. F., MCCARTHY, P. J.,1958: Sampling Opinion. An Analysis of Survey Procedure. New York, John Wiley and Sons.

STEVENS, D. L., 1997: Variable Density Grid-Based Sampling Designs for Continuous Spatial Populations. *Environmetrics*. 8. pp. 167-195.

STEVENS, D. L., OLSEN, A. R., 2004: Spatially Balanced Sampling of Natural Resources. *Journal of the American Statistical Association*. 99.465. pp. 262-278.

WANG, J., HANING, R., CAO, Z., 2010: Sample surveying to estimate the mean of a heterogeneous surface: reducing the error varinance through zoning. International Journal of Geographic Information Science. 24.4. pp. 523-543.

WANG, J., HAINING, R., CAO, Z., 2010: Sample surveying to estimate the mean of a heterogeneous surface: reducing the error variance through zoning. *International Journal of Geographical Information Science*. 24.4. pp. 523-543.

WANG, J., STEIN, A., GAO, B., GE, Y., 2012: A review of spatial sampling. *Spatial Statistics*. 2. pp. 1-14.

11 Melléklet

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 VetSamp
                                 A QGIS plugin
 Plugin for areal sampling based on GRTS
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/
                              -------------------
        begin                : 2019-09-26
        git sha              : $Format:%H$
        copyright            : (C) 2019 by Márton Papp
        email                : pappmarci95@gmail.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
import sys
import os
import tempfile
from PyQt5.QtCore import QSettings, QTranslator, qVersion, QCoreApplication, QVariant, Qt, QMetaType
from PyQt5.QtGui import QIcon

from PyQt5.QtWidgets import QAction, QDialog, QWidget, QPushButton, QTextBrowser, QLabel, QVBoxLayout, QHBoxLayout,
QLineEdit, QListWidget, QComboBox, QRadioButton, QButtonGroup, QScrollArea
from random import uniform, sample, randint, shuffle
from qgis.core import *
import processing
from qgis.analysis import QgsZonalStatistics
import math
import collections
import copy
import time
from decimal import *
import threading


# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .VetSamp_dialog import VetSampDialog
import os.path


class VetSamp:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'VetSamp_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&VetSamp')

        # Check if plugin was started the first time in current QGIS session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None



    # Creating feedback value for custom quadrant number selection, oversampling and stratified sampling:
        self.quadrchecker = 1
        self.is_oversampling = 1
        self.is_strat = 1


```

```
94
95              # STRATIFICATION DATA STORING VARIABLES:
96
97
            # Creating a list that stores the ids of the used fields (in lists of the stratification determiner id and
the field id):
98              self.usedfields = []
99
100
            # Creating value that stores stratum data (It is a list of lists containing the data of one stratification
determiner,
101             # The first value in the list is the serial number, the second is the field index, the third is type (1 if
a number, 2 if a discrete variable),
102             # And last follows the list of ranges and variables with their identification name (lists again) (ID, VALUE
/RANGE_MIN+RANGE_MAX))
103             # And if it is a contnuous data there is the maximum of the whole range at the end:
104             self.stratdata = []
105
106             # Creating an ID generator for the stratification:
107             self.id_numb = 0
108
109
            # Value for the selection between continuous and discrete type of field elements in the case of all number
attributes (1 is for continuous and 2 is for discrete values):
110             self.sel_type = 1
111
112
            # Creating a list for used identification names (A lsit of list. The first element is the serial number and
 the second is a list of used ids):
113             self.usedidnames = []
114
115
116
117             # LOCAL ASSIGMENT DATA STORING:
118
119
            # List for storing the id names inside an assignment cycle (A list of lists. Inside the list there is the s
erial of the id, and the id itself):
120             self.this_usedids = []
121
122             # Value for storing the current minimum of ranges:
123             self.current_min = float()
124
125
            # List to store current ranges, and the serial of the ranges (These are lists of the serial, the id and the
 value or the min and max of the range):
126             self.current_ranges = []
127             self.serial_for_ranges = 1
128
129             # List storing the used serials in range or discrete value id assignments:
130             self.used_serials = []
131
132
133
134             # Value for storing the qlistwidget for stratum determiner assignning:
135             self.stratlist = QListWidget()
136
137             # Creating the window for asking if stratification is desired:
138             self.windw_quest = QWidget()
139             self.windw_quest.setWindowTitle("Selection confirmation")
140             self.label = QLabel("Are you sure you want to use stratified sampling?")
141             self.yes_button = QPushButton("YES")
142             self.yes_button.clicked.connect(lambda: self.ok_for_strat())
143             self.no_button = QPushButton("NO")
144             self.no_button.clicked.connect(lambda: self.windw_quest.close())
145             self.hbox = QHBoxLayout()
146             self.hbox.addWidget(self.yes_button)
147             self.hbox.addWidget(self.no_button)
148             self.vbox = QVBoxLayout()
149             self.vbox.addWidget(self.label)
150             self.vbox.addLayout(self.hbox)
151             self.windw_quest.setLayout(self.vbox)
152
153
154             # Creating the list for the stratums in the case of stratified sampling:
155             #self.stratlist = QListWidget()
156
157             # Creating the list for the intervals for the definition of continuous variable stratums:
158             #self.listofranges = QListWidget()
159
160
161
162
163         # noinspection PyMethodMayBeStatic
164         def tr(self, message):
165             """Get the translation for a string using Qt translation API.
166
167             We implement this ourselves since we do not inherit QObject.
168
169             :param message: String for translation.
170             :type message: str, QString
171
172             :returns: Translated version of message.
173             :rtype: QString
174             """
175             # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
176             return QCoreApplication.translate('VetSamp', message)
177
178
179         def add_action(
180             self,
```

```
            icon_path,
            text,
            callback,
            enabled_flag=True,
            add_to_menu=True,
            add_to_toolbar=True,
            status_tip=None,
            whats_this=None,
            parent=None):
            """Add a toolbar icon to the toolbar.

            :param icon_path: Path to the icon for this action. Can be a resource
                path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
            :type icon_path: str

            :param text: Text that should be shown in menu items for this action.
            :type text: str

            :param callback: Function to be called when the action is triggered.
            :type callback: function

            :param enabled_flag: A flag indicating if the action should be enabled
                by default. Defaults to True.
            :type enabled_flag: bool

            :param add_to_menu: Flag indicating whether the action should also
                be added to the menu. Defaults to True.
            :type add_to_menu: bool

            :param add_to_toolbar: Flag indicating whether the action should also
                be added to the toolbar. Defaults to True.
            :type add_to_toolbar: bool

            :param status_tip: Optional text to show in a popup when mouse pointer
                hovers over the action.
            :type status_tip: str

            :param parent: Parent widget for the new action. Defaults None.
            :type parent: QWidget

            :param whats_this: Optional text to show in the status bar when the
                mouse pointer hovers over the action.

            :returns: The action that was created. Note that the action is also
                added to self.actions list.
            :rtype: QAction
            """

            icon = QIcon(icon_path)
            action = QAction(icon, text, parent)
            action.triggered.connect(callback)
            action.setEnabled(enabled_flag)

            if status_tip is not None:
                action.setStatusTip(status_tip)

            if whats_this is not None:
                action.setWhatsThis(whats_this)

            if add_to_toolbar:
                # Adds plugin icon to Plugins toolbar
                self.iface.addToolBarIcon(action)

            if add_to_menu:
                self.iface.addPluginToMenu(
                    self.menu,
                    action)

            self.actions.append(action)

            return action

        def initGui(self):
            """Create the menu entries and toolbar icons inside the QGIS GUI."""

            icon_path = ':/plugins/VetSamp/icon.png'
            self.add_action(
                icon_path,
                text=self.tr(u'Areal'),
                callback=self.run,
                parent=self.iface.mainWindow())

            # will be set False in run()
            self.first_start = True


        def unload(self):
            """Removes the plugin menu item and icon from QGIS GUI."""
            for action in self.actions:
                self.iface.removePluginMenu(
                    self.tr(u'&VetSamp'),
                    action)
                self.iface.removeToolBarIcon(action)



        # --------------------------- RASTER LAYER AVERAGING INSIDE POLYGONS ---------------------------------
-------------



```

```
283
284
285         # Error window for raster processing:
286         def errorwindow(self, numb, selected_vect_layer):
287             text = ""
288             if numb == 4:
289                 text = "There should be a raster layer loaded"
290             if numb == 7:
291                 text = "A layer must be selected"
292             if numb == 9:
293                 text = "The name of the raster average field must be given"
294             if numb == 10:
295
              text = "The number of the attribute selected should be below or equal to the number of attributes the l
ayer have"
296             windw = QWidget()
297             windw.setWindowTitle("Error window")
298             textbox = QTextBrowser()
299             textbox.append(text)
300             ok_button = QPushButton()
301             ok_button.setText("OK")
302             if numb == 4:
303                 ok_button.clicked.connect(lambda: self.errorclose(windw))
304             if numb == 7:
305                 ok_button.clicked.connect(lambda: self.raster(selected_vect_layer, windw, 1))
306             if numb == 9:
307                 ok_button.clicked.connect(lambda: self.raster(selected_vect_layer, windw, 1))
308             if numb == 10:
309                 ok_button.clicked.connect(lambda: windw.close())
310             vbox = QVBoxLayout()
311             vbox.addWidget(textbox)
312             vbox.addWidget(ok_button)
313             windw.setLayout(vbox)
314             windw.show()
315
316
317         # Function for error window to close itself and open self.raster():
318         def errorclose(self, windw):
319             windw.close()
320
321
322         # Function for selecting the raster layer:
323         def raster(self, selected_vect_layer, windw, is_windw):
324             if is_windw == 1:
325                 windw.close()
326             loaded_layers = self.iface.mapCanvas().layers()
327             raster_layers = []
328             for layer in loaded_layers:
329                 if layer.type() == QgsMapLayer.RasterLayer:
330                     raster_layers.append(layer)
331             loaded_layers = raster_layers
332             if not loaded_layers:
333                 self.errorwindow(4, selected_vect_layer)
334             else:
335                 loaded_layers_list = []
336                 for lay in loaded_layers:
337                     layname = lay.name()
338                     loaded_layers_list.append(layname)
339                 windw = QWidget()
340                 windw.setWindowTitle("Raster layer and attribute selection")
341                 label1 = QLabel()
342                 label1.setText("Raster layer selection:")
343                 label2 = QLabel()
344
              label2.setText("Name of the raster average field (It will be the given name and _mean in the field list
!):")
345                 rastlist = QComboBox()
346                 rastlist.clear()
347                 rastlist.addItems(loaded_layers_list)
348                 nameoffield = QLineEdit()
349                 ok_button = QPushButton()
350                 ok_button.setText("OK")
351                 ok_button.clicked.connect(
352
                   lambda: self.rasterlaycheck(windw, selected_vect_layer, rastlist.currentText(), nameoffield.text())
)
353                 vbox = QVBoxLayout()
354                 vbox.addWidget(label1)
355                 vbox.addWidget(rastlist)
356                 vbox.addWidget(label2)
357                 vbox.addWidget(nameoffield)
358                 vbox.addWidget(ok_button)
359                 windw.setLayout(vbox)
360                 windw.show()
361
362
363
        # Function for checking raster layer if it has a unique name and getting the selected attributes index:
364         def rasterlaycheck(self, windw, selected_vect_layer, selecteditems, nameoffield):
365             windw.close()
366             nameoffield = str(nameoffield)
367             if not nameoffield:
368                 self.errorwindow(9, selected_vect_layer)
369             selected_rast_layer_name = selecteditems
370             if not selected_rast_layer_name:
371                 self.errorwindow(7, selected_vect_layer)
372             else:
373                 selected_rast_layer = QgsProject.instance().mapLayersByName(selected_rast_layer_name)[0]
374                 self.rastertopolygon(selected_vect_layer, selected_rast_layer, nameoffield)
375
376
377         # Function to acquire the information for polygons and doing the averaging:
378         def rastertopolygon(self, selected_vect_layer, selected_rast_layer, nameoffield):
```

```
            nameoffield = str(nameoffield) + str("_")
            zonestat = QgsZonalStatistics(selected_vect_layer, selected_rast_layer, str(nameoffield), 1,
                                          QgsZonalStatistics.Mean)
            zonestat.calculateStatistics(None)



        # ----------------------------------------STRATIFIED SAMPLING-----------------------------------------------
---------------



        # Window for asking if truly stratified sampling is desired:
        def is_strat_sel(self):
            self.windw_quest.show()


        # Function for opening the stratum selection:
        def ok_for_strat(self):
            self.windw_quest.close()
            self.stratlaysel()


        # Function for selecting the layer for sampling:
        def stratlaysel(self):
            self.dlg.close()
            # Setting the stratum determiner fields' id generator number to zero:
            self.id_numb = 0
            # Setting the used fields storing list to an empty one:
            self.usedfields = []
            # Setting the list containing the already used ids to an empty one:
            self.usedidnames = []
            # Setting the stratum determiner data storing list to empty:
            self.stratdata = []
            windw = QWidget()
            windw.setWindowTitle("Layer selection")
            label = QLabel("PLease select the layer for sampling:")
            cbox = QComboBox()
            lays = self.iface.mapCanvas().layers()
            vect_lays = []
            for layer in lays:
                if layer.type() == QgsMapLayer.VectorLayer:
                    layer_name = layer.name()
                    vect_lays.append(layer_name)
            cbox.addItems(vect_lays)
            ok_button = QPushButton("OK")
            ok_button.clicked.connect(lambda: self.usestrat(windw, cbox.currentText()))
            cancel_button = QPushButton("CANCEL")
            cancel_button.clicked.connect(lambda: windw.close())
            hbox = QHBoxLayout()
            hbox.addWidget(ok_button)
            hbox.addWidget(cancel_button)
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            vbox.addWidget(cbox)
            vbox.addLayout(hbox)
            windw.setLayout(vbox)
            windw.show()


        # Function for the settings of stratified sampling:
        def usestrat(self, windw, layname):
            windw.close()
            layer = QgsProject.instance().mapLayersByName(layname)[0]
            windw = QWidget()
            windw.setWindowTitle("Stratification settings")

        label1 = QLabel("If raster point averaging inside polygons is needed, please press AVERAGE RASTER button:")

            rast_button = QPushButton("AVERAGE RASTER")
            rast_button.clicked.connect(lambda: self.raster(layer, 0, 0))
            self.stratlist = QListWidget()
            self.stratlist.clear()
            plus_button = QPushButton("+")
            plus_button.clicked.connect(lambda: self.stratspec(layer))
            minus_button = QPushButton("-")
            minus_button.clicked.connect(lambda: self.delstratspec(self.stratlist.selectedItems()))
            info_button = QPushButton("INFO")
            info_button.clicked.connect(lambda: self.infostratspec(self.stratlist.selectedItems(), layer))
            ok_button = QPushButton("OK")
            ok_button.clicked.connect(lambda: self.sampperstratum(windw, layer))
            cancel_button = QPushButton("CANCEL")
            cancel_button.clicked.connect(lambda: windw.close())
            hbox1 = QHBoxLayout()
            hbox1.addWidget(label1)
            hbox1.addWidget(rast_button)
            vbox = QVBoxLayout()
            vbox.addLayout(hbox1)
            vbox.addWidget(self.stratlist)
            hbox2 = QHBoxLayout()
            hbox2.addWidget(plus_button)
            hbox2.addWidget(minus_button)
            hbox2.addWidget(info_button)
            vbox.addLayout(hbox2)
            hbox3 = QHBoxLayout()
            hbox3.addWidget(ok_button)
            hbox3.addWidget(cancel_button)
            vbox.addLayout(hbox3)
            windw.setLayout(vbox)
```

```
            windw.show()


        # Function for adding new stratum determiner:
        def stratspec(self, layer):
            windw = QWidget()
            windw.setWindowTitle("Stratum determiner selection")
            label = QLabel("Please select the field to use:")
            cbox = QComboBox()
            field_names = []
            fields = layer.fields()
            usedfields = []
            for a in self.usedfields:
                usedfields.append(a[1])
            for field in fields:
                field_index = layer.fields().indexFromName(field.name())
                if field_index in usedfields:
                    pass
                else:
                    field_names.append(field.name())
            cbox.addItems(field_names)
            ok_button = QPushButton("OK")
            ok_button.clicked.connect(lambda: self.getfielddata(cbox.currentText(), windw, layer))
            cancel_button = QPushButton("CANCEL")
            cancel_button.clicked.connect(lambda: windw.close())
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            vbox.addWidget(cbox)
            vbox.addWidget(ok_button)
            vbox.addWidget(cancel_button)
            windw.setLayout(vbox)
            windw.show()


    # Function for adding the id and field id for the list containing tha data of the determiner and getting the ty
pe of it(whether it is a discrete variable or a continuous):
        def getfielddata(self, field, windw, layer):
            windw.close()
            # Checking if there is a field selected:
            is_error = 0
            if not field:
                is_error = is_error + 1
            if is_error != 0:
                self.error_for_field()
            else:
                # Creating a value that is 1 if no non-number element is found and 2 if there is any:
                is_there_non_number = 1
                field_id = layer.fields().indexFromName(str(field))
                field_data_list = [field_id]

             # Creating a list of the fields elements and simultaneously checking whether these elements are numbers
 or only usables as discrete values:
                feats = layer.getFeatures()
                field_elems = []
                break_for_null = 0
                for feat in feats:
                    attrs = feat.attributes()
                    field_elem = attrs[field_id]
                    if not field_elem:
                        break_for_null = 1
                        break
                    else:
                        field_elem = str(field_elem)
                        if is_there_non_number == 1:
                            field_elem_is_numb = self.is_numb(field_elem)
                            if field_elem_is_numb == False:
                                is_there_non_number = 2
                        field_elems.append(field_elem)
                if break_for_null == 1:
                    self.error_for_NULL()
                else:
                    if is_there_non_number == 2:
                        field_data_list.append(is_there_non_number)
                        self.val_ids(field_data_list, field_elems, field, layer)
                    else:
                        self.disc_or_cont(field_data_list, field_elems, field, layer)


        # function for checking if a value is a number:
        def is_numb(self, value):
            try:
                float(value)
                return True
            except ValueError:
                return False


        # Error window for field selection:
        def error_for_field(self):
            windw = QWidget()
            windw.setWindowTitle("Error window")
            label = QLabel("A field must be selected, please make sure that your layer does have fields!")
            cancel_button = QPushButton("CANCEL")
            cancel_button.clicked.connect(lambda: windw.close())
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            vbox.addWidget(cancel_button)
            windw.setLayout(vbox)
            windw.show()


        # Function for error window if there is a NULL field value:
```

```
578         def error_for_NULL(self):
579             windw = QWidget()
580             windw.setWindowTitle("Error_window")
581             label = QLabel("Please use a field that does not contain NULL values!")
582             ok_button = QPushButton("OK")
583             ok_button.clicked.connect(lambda: windw.close())
584             hbox = QHBoxLayout()
585             hbox.addStretch(2)
586             hbox.addWidget(ok_button)
587             hbox.addStretch(2)
588             vbox = QVBoxLayout()
589             vbox.addWidget(label)
590             vbox.addLayout(hbox)
591             windw.setLayout(vbox)
592             windw.show()
593 
594 
595         # Function for selecting between discrete and continuous values:
596         def disc_or_cont(self, field_data_list, field_elems, field, layer):
597             # Setting the value for the type selected (1 is continuous and 2 is discrete):
598             self.sel_type = 1
599             windw = QWidget()
600             windw.setWindowTitle("value type selection")
601             label = QLabel("Please select the correct value type for stratum determining field:")
602             label1 = QLabel("Use values as continuous data:")
603             label2 = QLabel("Use values as discrete data")
604             button1 = QRadioButton()
605             button1.setChecked(True)
606             button1.toggled.connect(lambda: self.cont_but())
607             button2 = QRadioButton()
608             button2.toggled.connect(lambda: self.disc_but())
609             ok_button = QPushButton("OK")
610             ok_button.clicked.connect(
611                 lambda: self.use_cont_or_disc(field_data_list, field_elems, windw, field, layer))
612             cancel_button = QPushButton("CANCEL")
613             cancel_button.clicked.connect(lambda: windw.close())
614             hbox = QHBoxLayout()
615             hbox.addWidget(label1)
616             hbox.addWidget(button1)
617             hbox.addStretch(5)
618             hbox.addWidget(label2)
619             hbox.addWidget(button2)
620             hbox2 = QHBoxLayout()
621             hbox2.addStretch(15)
622             hbox2.addWidget(ok_button)
623             hbox2.addWidget(cancel_button)
624             vbox = QVBoxLayout()
625             vbox.addWidget(label)
626             vbox.addLayout(hbox)
627             vbox.addLayout(hbox2)
628             windw.setLayout(vbox)
629             windw.show()
630 
631 
632         # Function for the action of continuous value radio button:
633         def cont_but(self):
634             self.sel_type = 1
635 
636 
637         # Function for the action of discrete value radio button:
638         def disc_but(self):
639             self.sel_type = 2
640 
641 
642 
    # Function for selecting the id generation of field elements or ranges based on the previous selection:
643         def use_cont_or_disc(self, field_data_list, field_elems, windw, field, layer):
644             windw.close()
645             if self.sel_type == 1:
646                 field_data_list.append(self.sel_type)
647                 self.range_ids(field_data_list, field_elems, field, layer)
648             if self.sel_type == 2:
649                 field_data_list.append(self.sel_type)
650                 self.val_ids(field_data_list, field_elems, field, layer)
651 
652 
653         # Function for generating ids for ranges of the field values (when they are used as continuous types):
654         def range_ids(self, field_data_list, field_elems, field, layer):
655             field_elems = [float(i) for i in field_elems]
656             whole_field_max = max(field_elems)
657             self.current_min = min(field_elems)
658             whole_field_min = copy.deepcopy(self.current_min)
659             self.serial_for_ranges = 1
660             self.this_usedids = []
661             self.current_ranges = []
662             self.used_serials = []
663             windw = QWidget()
664             windw.setWindowTitle("Range selection for continuous values")
665             self.listofranges = QListWidget()
666             self.listofranges.clear()
667             plus_button = QPushButton("+")
668             plus_button.clicked.connect(lambda: self.add_range(whole_field_max))
669             minus_button = QPushButton("-")
670             minus_button.clicked.connect(
671                 lambda: self.del_range(self.listofranges.selectedItems()[0], whole_field_min))
672             ok_button = QPushButton("OK")
673             ok_button.clicked.connect(
674                 lambda: self.finish_range(windw, field_data_list, field, layer, whole_field_max))
675             cancel_button = QPushButton("CANCEL")
676             cancel_button.clicked.connect(lambda: windw.close())
677             hbox1 = QHBoxLayout()
678             hbox1.addWidget(plus_button)
679             hbox1.addWidget(minus_button)
```

```
            hbox2 = QHBoxLayout()
            hbox2.addWidget(ok_button)
            hbox2.addWidget(cancel_button)
            vbox = QVBoxLayout()
            vbox.addWidget(self.listofranges)
            vbox.addLayout(hbox1)
            vbox.addLayout(hbox2)
            windw.setLayout(vbox)
            windw.show()

        # Function for adding new range:
        def add_range(self, whole_field_max):
            if float(self.current_min) == float(whole_field_max):
                window_er = QWidget()
                window_er.setWindowTitle("Error window")

             label_er = QLabel("The last range is already assigned, please proceed to the finalization of these rang
es!")
                ok_button_er = QPushButton("OK")
                ok_button_er.clicked.connect(lambda: window_er.close())
                vbox_er = QVBoxLayout()
                vbox_er.addWidget(label_er)
                hbox = QHBoxLayout()
                hbox.addStretch(5)
                hbox.addWidget(ok_button_er)
                hbox.addStretch(5)
                vbox_er.addLayout(hbox)
                window_er.setLayout(vbox_er)
                window_er.show()
            else:
                windw = QWidget()
                windw.setWindowTitle("Range settings")
                label = QLabel(

                 "Please select the range and enter an ID:" + "\n" + "(Note that the maximum of the values in the se
lected field is : " + str(

                  whole_field_max) + "\nAnd also note that the assigned upper value of the range is not included[
that is the begining of the next range]\nexcept for the last one)")
                label1 = QLabel("Range: " + str(self.current_min) + "- ")
                newmax = QLineEdit()
                label2 = QLabel("ID: ")
                this_id = QLineEdit()
                ok_button = QPushButton("OK")
                ok_button.clicked.connect(
                    lambda: self.assign_range(whole_field_max, newmax.text(), this_id.text(), windw))
                cancel_button = QPushButton("CANCEL")
                cancel_button.clicked.connect(lambda: windw.close())
                hbox1 = QHBoxLayout()
                hbox1.addWidget(label1)
                hbox1.addWidget(newmax)
                hbox2 = QHBoxLayout()
                hbox2.addWidget(label2)
                hbox2.addWidget(this_id)
                hbox3 = QHBoxLayout()
                hbox3.addWidget(ok_button)
                hbox3.addWidget(cancel_button)
                vbox = QVBoxLayout()
                vbox.addWidget(label)
                vbox.addLayout(hbox1)
                vbox.addLayout(hbox2)
                vbox.addLayout(hbox3)
                windw.setLayout(vbox)
                windw.show()

        # Function for checking the previous settings and if correct assign the values:
        def assign_range(self, whole_field_max, newmax, this_id, windw):
            windw.close()
            newmax = float(newmax)
            error_text = []
            is_error = 0
            is_numb_max = self.is_numb(newmax)
            if not newmax:
                is_error = is_error + 1
                error_text.append("A range maximum must be given!")
            else:
                if is_numb_max == False:
                    error_text.append("The given maximum must be a number!")
                    is_error = is_error + 1
                if is_numb_max == True:
                    if newmax > float(whole_field_max):
                        is_error = is_error + 1
                        error_text.append("The last range can not exceed the maximum of the field values!")
                    if newmax < float(self.current_min):
                        is_error = is_error + 1
                        error_text.append("The maximum of the range must be over the given minimum!")
            if not this_id:
                is_error = is_error + 1
                error_text.append("A range ID must be given!")
            is_repeated_id = 0
            for elem in self.usedidnames:
                if this_id in elem[1]:
                    is_repeated_id = is_repeated_id + 1
            usedids = []
            for ids in self.this_usedids:
                usedids.append(ids[1])
            if this_id in usedids:
                is_repeated_id = is_repeated_id + 1
            if is_repeated_id != 0:
                is_error = is_error + 1
                error_text.append("The ID must be unique, already used IDs are not accepted!")
```

```
            if is_error != 0:
                self.error_range(error_text, whole_field_max)
            else:
                usedids = [self.serial_for_ranges, this_id]
                self.this_usedids.append(usedids)
                this_range = [self.serial_for_ranges, this_id, self.current_min, newmax]
                self.current_ranges.append(this_range)
                this_range = ', '.join(map(str, this_range))
                self.listofranges.addItem(this_range)
                self.used_serials.append(self.serial_for_ranges)
                self.current_min = newmax
                self.serial_for_ranges = self.serial_for_ranges + 1

    # Function for error window for range assignment:
    def error_range(self, error_text, whole_field_max):
        windw = QWidget()
        windw.setWindowTitle("Error window")
        label = QLabel("The following error(s) occurred:")
        textbox = QTextBrowser()
        error_text = '\n'.join(error_text)
        textbox.append(error_text)
        ok_button = QPushButton("OK")
        ok_button.clicked.connect(lambda: self.reopen_range(windw, whole_field_max))
        cancel_button = QPushButton("CANCEL")
        cancel_button.clicked.connect(lambda: windw.close())
        hbox = QHBoxLayout()
        hbox.addWidget(ok_button)
        hbox.addWidget(cancel_button)
        vbox = QVBoxLayout()
        vbox.addWidget(label)
        vbox.addWidget(textbox)
        vbox.addLayout(hbox)
        windw.setLayout(vbox)
        windw.show()

    # Function for reopening range selection:
    def reopen_range(self, windw, whole_field_max):
        windw.close()
        self.add_range(whole_field_max)

    # Function for deleting assigned ranges:
    def del_range(self, item, whole_field_min):
        if not item:
            windw = QWidget()
            windw.setWindowTitle("Error window")
            label = QLabel("Please select a range to delete!")
            cancel_button = QPushButton("CANCEL")
            cancel_button.clicked.connect(lambda: windw.close())
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            vbox.addWidget(cancel_button)
            windw.setLayout(vbox)
            windw.show()
        else:
            item = item.text()
            item_serial = str()
            for a in item:
                if a != ",":
                    item_serial = item_serial + str(a)
                else:
                    break
            item_serial = int(item_serial)
            serials_to_del = []
            listelems = [self.listofranges.item(i) for i in range(self.listofranges.count())]
            for x in self.used_serials:
                if item_serial <= x:
                    serials_to_del.append(x)
            for y in serials_to_del:
                for ids in self.this_usedids:
                    if ids[0] == y:
                        ind = self.this_usedids.index(ids)
                        del self.this_usedids[ind]
                for rngs in self.current_ranges:
                    if rngs[0] == y:
                        ind = self.current_ranges.index(rngs)
                        del self.current_ranges[ind]
                indofser = self.used_serials.index(y)
                del self.used_serials[indofser]
                for elem in listelems:
                    elem_text = elem.text()
                    elem_ser = str()
                    for b in elem_text:
                        if b != ",":
                            elem_ser = elem_ser + str(b)
                        else:
                            break
                    if int(elem_ser) == int(y):
                        self.listofranges.takeItem(self.listofranges.row(elem))
            current_maxes = [float(whole_field_min)]
            for elem in self.current_ranges:
                current_maxes.append(elem[3])
            newmax = max(current_maxes)
            self.current_min = newmax

    # Function for finishing the assigned ranges:
    def finish_range(self, windw, field_data_list, field, layer, whole_field_max):
        if float(self.current_min) != float(whole_field_max):
            erwind = QWidget()
            erwind.setWindowTitle("Error window")
```

```
                label = QLabel(

                 "Please continue the range selection until the last is assigned (when you reach the maximum of valu
es)!")
                ok_button = QPushButton("OK")
                ok_button.clicked.connect(lambda: erwind.close())
                vbox = QVBoxLayout()
                vbox.addWidget(label)
                vbox.addWidget(ok_button)
                erwind.setLayout(vbox)
                erwind.show()
            else:
                windw.close()
                appendlist = []
                for ids in self.this_usedids:
                    thislist = [ids[1]]
                    for elem in self.current_ranges:
                        if elem[1] == ids[1]:
                            thislist.append(elem[2])
                            thislist.append(elem[3])
                    appendlist.append(thislist)
                field_data_list.append(appendlist)
                self.id_numb = self.id_numb + 1
                field_data_list.insert(0, self.id_numb)
                field_data_list.append(whole_field_max)
                self.stratdata.append(field_data_list)
                newusedids = [self.id_numb]
                idlist = []
                for ids in self.this_usedids:
                    idlist.append(ids[1])
                newusedids.append(idlist)
                self.usedidnames.append(newusedids)
                list_to_stratlist = [self.id_numb, field, "Continuous data type", whole_field_max]
                list_to_stratlist = ', '.join(map(str, list_to_stratlist))
                self.stratlist.addItem(list_to_stratlist)
                field_id = layer.fields().indexFromName(field)
                self.usedfields.append([self.id_numb, field_id])


        # Function for ID assignment window for discrete field values:
        def val_ids(self, field_data_list, field_elems, field, layer):
            windw = QWidget()
            windw.setWindowTitle("ID selection")
            vbox = QVBoxLayout()
            scrollbox = QScrollArea()
            vbox.addWidget(scrollbox)
            scrollbox.setWidgetResizable(True)
            scrollarea = QWidget(scrollbox)
            scrollvbox = QVBoxLayout(scrollarea)
            scrollarea.setLayout(scrollvbox)
            label_dict = {}
            line_edit_dict = {}
            hbox_dict = {}
            elem_count = 1

        # Creating a list to store the used elements with their serial and element name, to use it for the id and v
alue assignment:
            # It is a list of lists containing the serial number and the elem name
            usedelems = []

        # A list containing just the name of the used elements to know if they are already got an ID selection line
:
            usedelems2 = []
            for elem in field_elems:
                if elem in usedelems2:
                    pass
                else:

                 label_dict["label" + str(elem_count)] = QLabel("Please enter the ID of the field elem: " + str(elem
))
                    line_edit_dict["line" + str(elem_count)] = QLineEdit()
                    hbox_dict["hbox" + str(elem_count)] = QHBoxLayout()
                    thislabel = label_dict.get("label" + str(elem_count))
                    thisline = line_edit_dict.get("line" + str(elem_count))
                    thishbox = hbox_dict.get("hbox" + str(elem_count))
                    thishbox.addWidget(thislabel)
                    thishbox.addWidget(thisline)
                    scrollvbox.addLayout(thishbox)
                    usedelems.append([elem_count, elem])
                    usedelems2.append(elem)
                    elem_count = elem_count + 1
            scrollbox.setWidget(scrollarea)
            ok_button = QPushButton("OK")
            ok_button.clicked.connect(
                lambda: self.check_disc_ids(field_data_list, field_elems, field, layer, windw, line_edit_dict,
                                            usedelems, usedelems2))
            cancel_button = QPushButton("CANCEL")
            cancel_button.clicked.connect(lambda: windw.close())
            hbox = QHBoxLayout()
            hbox.addWidget(ok_button)
            hbox.addWidget(cancel_button)
            vbox.addLayout(hbox)
            windw.setLayout(vbox)
            windw.show()


        # Function for getting and checking assignment:

    def check_disc_ids(self, field_data_list, field_elems, field, layer, windw, line_edit_dict, usedelems, usedelem
s2):
            windw.close()
            is_error = 0
            is_missing_value = 0
```

```
            missinglist = []
            is_used_value = 0
            usedlist = []
            for elem in usedelems:
                counter = elem[0]
                thisline_er = line_edit_dict.get("line" + str(counter))
                thisid_er = thisline_er.text()
                if not thisid_er:
                    is_missing_value = is_missing_value + 1
                    is_error = is_error + 1
                    missinglist.append(elem)
                else:
                    if thisid_er in self.usedidnames or thisid_er in usedelems2:
                        is_error = is_error + 1
                        is_used_value = is_used_value + 1
                        usedlist.append(elem)
            if is_error != 0:
                windw = QWidget()
                windw.setWindowTitle("Error window")
                label = QLabel("The following error(s) occured:")
                vbox = QVBoxLayout()
                vbox.addWidget(label)
                if is_missing_value != 0:
                    label2 = QLabel("The following field element(s) doesn't have an ID assigned:")
                    textbox = QTextBrowser()
                    missinglist = ', '.join(map(str, missinglist))
                    textbox.append(missinglist)
                    vbox.addWidget(label2)
                    vbox.addWidget(textbox)
                if is_used_value != 0:
                    label3 = QLabel("The following field element(s) got an ID which was already used:")
                    textbox2 = QTextBrowser()
                    usedlist = ', '.join(map(str, usedlist))
                    textbox2.append(usedlist)
                    vbox.addWidget(label3)
                    vbox.addWidget(textbox2)
                ok_button = QPushButton("OK")
                ok_button.clicked.connect(
                    lambda: self.ok_repoen_disc(field_data_list, field_elems, field, layer, windw))
                cancel_button = QPushButton("CANCEL")
                cancel_button.clicked.connect(lambda: windw.close())
                hbox = QHBoxLayout()
                hbox.addWidget(ok_button)
                hbox.addWidget(cancel_button)
                vbox.addLayout(hbox)
                windw.setLayout(vbox)
                windw.show()
            else:
                ids_list = []
                here_used_ids = []
                for elem in usedelems:
                    thisline = line_edit_dict.get("line" + str(elem[0]))
                    thisid = thisline.text()
                    thislist = [thisid, elem[1]]
                    ids_list.append(thislist)
                    here_used_ids.append(thisid)
                field_data_list.append(ids_list)
                self.id_numb = self.id_numb + 1
                field_data_list.insert(0, self.id_numb)
                self.stratdata.append(field_data_list)
                newusedids = [self.id_numb]
                newusedids.append(here_used_ids)
                self.usedidnames.append(newusedids)
                list_to_stratlist = [self.id_numb, field, "Discrete data type"]
                list_to_stratlist = ', '.join(map(str, list_to_stratlist))
                self.stratlist.addItem(list_to_stratlist)
                field_id = layer.fields().indexFromName(field)
                self.usedfields.append([self.id_numb, field_id])


    # Function for reopen the id assignment window for discrete values if an error was found and ok is pressed:
        def ok_repoen_disc(self, field_data_list, field_elems, field, layer, windw):
            windw.close()
            self.val_ids(field_data_list, field_elems, field, layer)


        # Function for deleting stratification determiner:
        def delstratspec(self, stratdet):
            stratdet = stratdet[0].text()
            if not stratdet:
                windw = QWidget()
                windw.setWindowTitle("Error window")
                label = QLabel("Please select a stratification determiner!")
                ok_button = QPushButton("OK")
                ok_button.clicked.connect(lambda: windw.close())
                hbox = QHBoxLayout()
                hbox.addStretch(2)
                hbox.addWidget(ok_button)
                hbox.addStretch(2)
                vbox = QVBoxLayout()
                vbox.addWidget(label)
                vbox.addLayout(hbox)
                windw.setLayout(vbox)
                windw.show()
            else:
                indnumb = str()
                for ltr in stratdet:
                    if ltr != ",":
                        indnumb = indnumb + str(ltr)
                    else:
                        break
                strat_id = int(indnumb)
```

```
                for elem in self.usedfields:
                    if int(elem[0]) == strat_id:
                        ind = self.usedfields.index(elem)
                        del self.usedfields[ind]
                for elem in self.usedidnames:
                    if int(elem[0]) == strat_id:
                        ind = self.usedidnames.index(elem)
                        del self.usedidnames[ind]
                for elem in self.stratdata:
                    if int(elem[0]) == strat_id:
                        ind = self.stratdata.index(elem)
                        del self.stratdata[ind]
                all_stratdet = [self.stratlist.item(i) for i in range(self.stratlist.count())]
                for elem in all_stratdet:
                    elem_text = elem.text()
                    elem_ser = str()
                    for b in elem_text:
                        if b != ",":
                            elem_ser = elem_ser + str(b)
                        else:
                            break
                    if int(elem_ser) == strat_id:
                        self.stratlist.takeItem(self.stratlist.row(elem))


        # Function for getting the information about the selected stratification determiner:
        def infostratspec(self, spec, layer):
            spec = spec[0].text()
            indnumb = str()
            for ltr in spec:
                if ltr != ",":
                    indnumb = indnumb + str(ltr)
                else:
                    break
            for tlist in self.stratdata:
                if int(tlist[0]) == int(indnumb):
                    speclist = tlist
            fields = layer.fields()
            field_index = int(speclist[1])
            field_name = fields[field_index].name()
            id_value = str()
            if speclist[2] == 1:
                discrete_or_continuous = "Continuous variable"
                for elem in speclist[3]:
                    id_value = id_value + str(elem[0]) + ": " + str(elem[1]) + "-" + str(elem[2]) + "\n"
            else:
                discrete_or_continuous = "Discrete variable"
                for elem in speclist[3]:
                    id_value = id_value + str(elem[0]) + ": " + str(elem[1]) + "\n"
            windw = QWidget()
            windw.setWindowTitle("Information window")
            label = QLabel("Stratum determiner information:")
            textbox = QTextBrowser()

        appendtext = "ID: " + str(speclist[0]) + "\n" + "FIELD: " + str(field_name) + "\n" + "TYPE: " + str(
                discrete_or_continuous) + "\n" + "ID/VALUE PAIRS: " + "\n" + id_value
            if speclist[2] == 1:
                appendtext = appendtext + "MAXIMUM OF THE WHOLE RANGE: " + str(speclist[4])
            textbox.append(appendtext)
            ok_button = QPushButton("OK")
            ok_button.clicked.connect(lambda: windw.close())
            hbox = QHBoxLayout()
            hbox.addStretch(2)
            hbox.addWidget(ok_button)
            hbox.addStretch(2)
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            vbox.addWidget(textbox)
            vbox.addLayout(hbox)
            windw.setLayout(vbox)
            windw.show()


        # Function for error window for time out of stratum name generation:
        def error_timeout(self):
            windw = QWidget()
            windw.setWindowTitle("Error window")

        label = QLabel("Please make sure that you don't have too much fields with tha name stratum_name + a number!
")
            ok_button = QPushButton("OK")
            ok_button.clicked.connect(lambda: windw.close())
            hbox = QHBoxLayout()
            hbox.addStretch(2)
            hbox.addWidget(ok_button)
            hbox.addStretch(2)
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            vbox.addLayout(hbox)
            windw.setLayout(vbox)
            windw.show()


        # Function for sorting the stratum determiner list for stratum generation (by the field ID value):
        def sort_for_strat(self, elem):
            return elem[1]


        # Function for error window if no stratum determiner is given:
        def error_for_strat(self):
            windw = QWidget()
            windw.setWindowTitle("Error window")
            label = QLabel("Please add a stratum determiner before proceed to sampling!")
```

```
1175            ok_button = QPushButton("OK")
1176            ok_button.clicked.connect(lambda: windw.close())
1177            vbox = QVBoxLayout()
1178            vbox.addWidget(label)
1179            vbox.addWidget(ok_button)
1180            windw.setLayout(vbox)
1181            windw.show()
1182
1183
1184        # Function for selecting sample amount of each stratum:
1185        def sampperstratum(self, windw, layer):
1186            if not self.stratdata:
1187                self.error_for_strat()
1188            else:
1189                field_names = []
1190                for f in layer.fields():
1191                    field_names.append(f.name())
1192                probenumb = 1
1193                id_error = 0
1194                timeout = time.time() + 60
1195                stratname_index = int()
1196                while True:
1197                    stratname = "str_nm" + str(probenumb)
1198                    if stratname not in field_names:
1199                        layer.dataProvider().addAttributes([QgsField(stratname, QVariant.String)])
1200                        layer.updateFields()
1201                        stratname_index = layer.fields().indexFromName(stratname)
1202                        break
1203                    if time.time() > timeout:
1204                        id_error = 1
1205                        break
1206                    else:
1207                        probenumb = probenumb + 1
1208                if id_error == 1:
1209                    self.error_timeout()
1210                else:
1211                    windw.close()
1212                    stratdata = sorted(self.stratdata, key=self.sort_for_strat)
1213                    fields = layer.fields()
1214                    first_strat = 1
1215                    layer.startEditing()
1216                    for field in fields:
1217                        field_id = layer.fields().indexFromName(field.name())
1218                        for elem in stratdata:
1219                            if int(elem[1]) == int(field_id):
1220                                feats = layer.getFeatures()
1221                                for feat in feats:
1222                                    attrs = feat.attributes()
1223                                    value = attrs[field_id]
1224                                    if elem[2] == 1:
1225                                        for rang in elem[3]:
1226
                                            if float(rang[1]) <= float(value) < float(rang[2]) or float(value) == float
(
1227                                                rang[2]) == float(elem[4]):
1228                                            if first_strat == 1:
1229                                                changeto = str(rang[0])
1230                                            else:
1231                                                changeto = str(attrs[stratname_index]) + str(rang[0])
1232
                                        layer.changeAttributeValue(feat.id(), stratname_index, str(changeto))
1233                                        break
1234                                    if elem[2] == 2:
1235                                        for val in elem[3]:
1236                                            if str(value) == str(val[1]):
1237                                                if first_strat == 1:
1238                                                    changeto = str(val[0])
1239                                                else:
1240                                                    changeto = str(attrs[stratname_index]) + str(val[0])
1241
                                                layer.changeAttributeValue(feat.id(), stratname_index, str(changeto))
1242                                                break
1243                                    first_strat = first_strat + 1
1244                                    this_ind = stratdata.index(elem)
1245                                    del stratdata[this_ind]
1246                                    break
1247                    layer.commitChanges()
1248                    self.strat_is_over(layer, stratname_index)
1249
1250
1251        # Function for selecting between using and not using oversampling:
1252        def strat_is_over(self, layer, stratname_index):
1253            windw = QWidget()
1254            windw.setWindowTitle("Oversampling selection")
1255            label = QLabel(
1256
             "If oversampling is desired in each stratum, please select OVERSAMPLING button,\nif not, please select
REGULAR button)")
1257            oversample_button = QPushButton("OVERSAMPLING")
1258            oversample_button.clicked.connect(lambda: self.strat_samp(windw, 1, layer, stratname_index))
1259            regular_button = QPushButton("REGULAR")
1260            regular_button.clicked.connect(lambda: self.strat_samp(windw, 2, layer, stratname_index))
1261            hbox = QHBoxLayout()
1262            hbox.addWidget(oversample_button)
1263            hbox.addWidget(regular_button)
1264            vbox = QVBoxLayout()
1265            vbox.addWidget(label)
1266            vbox.addLayout(hbox)
1267            windw.setLayout(vbox)
1268            windw.show()
1269
1270
1271        # Function for opening the stratum sampling settings window:
```

```
    def strat_samp(self, windw_prev, selval, layer, stratname_index):
        windw_prev.close()
        strat_info = [int(selval)]
        strats = []
        feats = layer.getFeatures()
        for feat in feats:
            attrs = feat.attributes()
            this_strat = attrs[stratname_index]
            if this_strat in strats:
                pass
            else:
                strats.append(this_strat)

    # list for storing lists of stratums and their IDs (for retrieving information from qlineedit storing dictionaries [ID, stratum])
        strat_samp_data = []
        strat_dict = {}
        strat_line_edit_dict = {}
        strat_hbox_dict = {}
        strat_over_label = {}
        strat_over_line = {}
        strat_over_hbox = {}
        strat_over_vbox = {}
        windw = QWidget()
        windw.setWindowTitle("Stratum sampling settings")
        label = QLabel("Please enter the sample (and oversample) number for each stratum:")
        vbox = QVBoxLayout()
        vbox.addWidget(label)
        scrollbox = QScrollArea()
        vbox.addWidget(scrollbox)
        scrollbox.setWidgetResizable(True)
        scrollarea = QWidget(scrollbox)
        scrollvbox = QVBoxLayout(scrollarea)
        scrollarea.setLayout(scrollvbox)
        counter = 1
        for strat in strats:
            this_data = [int(counter), strat]
            strat_samp_data.append(this_data)
            strat_dict["stratum" + str(counter)] = QLabel(
                "Please enter the number of desired sample for the stratum: " + str(strat))
            strat_line_edit_dict["stratum" + str(counter)] = QLineEdit()
            strat_hbox_dict["hbox" + str(counter)] = QHBoxLayout()
            thislabel = strat_dict.get("stratum" + str(counter))
            thisline = strat_line_edit_dict.get("stratum" + str(counter))
            thishbox = strat_hbox_dict.get("hbox" + str(counter))
            thishbox.addWidget(thislabel)
            thishbox.addWidget(thisline)
            if selval == 1:
                strat_over_label["stratum" + str(counter)] = QLabel("Oversample number:")
                strat_over_line["stratum" + str(counter)] = QLineEdit()
                strat_over_hbox["stratum_hbox" + str(counter)] = QHBoxLayout()
                strat_over_vbox["stratum_vbox" + str(counter)] = QVBoxLayout()
                overlabel = strat_over_label.get("stratum" + str(counter))
                overline = strat_over_line.get("stratum" + str(counter))
                overhbox = strat_over_hbox.get("stratum_hbox" + str(counter))
                overvbox = strat_over_vbox.get("stratum_vbox" + str(counter))
                overhbox.addWidget(overlabel)
                overhbox.addWidget(overline)
                overvbox.addLayout(thishbox)
                overvbox.addLayout(overhbox)
                scrollvbox.addLayout(overvbox)
            else:
                scrollvbox.addLayout(thishbox)
            counter = counter + 1
        scrollbox.setWidget(scrollarea)
        ok_button = QPushButton("OK")
        ok_button.clicked.connect(

            lambda: self.finish_strat(strat_samp_data, strat_line_edit_dict, strat_over_line, selval, strat_info, layer,
                                      stratname_index, windw))
        cancel_button = QPushButton("CANCEL")
        cancel_button.clicked.connect(lambda: windw.close())
        hbox_button = QHBoxLayout()
        hbox_button.addWidget(ok_button)
        hbox_button.addWidget(cancel_button)
        vbox.addLayout(hbox_button)
        windw.setLayout(vbox)
        windw.show()


    # Function for checking the assigned sample and oversample numbers and then creating the final list storing them:
    def finish_strat(self, strat_samp_data, strat_line_edit_dict, strat_over_line, selval, strat_info, layer,
                     stratname_index, windw_prev):
        windw_prev.close()
        is_error = 0
        strat_list_to_append = []
        is_samp = 0
        is_samp_int = 0
        is_samp_b_zero = 0
        is_oversamp = 0
        is_oversamp_int = 0
        is_oversamp_b_zero = 0
        for elem in strat_samp_data:
            thislist = []
            this_numb = strat_line_edit_dict.get("stratum" + str(elem[0]))
            this_numb = this_numb.text()
            if not this_numb:
                is_samp = is_samp + 1
                is_error = is_error + 1
```

```
                else:
                    isint = self.integer_check(this_numb)
                    if isint == 1:
                        is_samp_int = is_samp_int + 1
                        is_error = is_error + 1
                    else:
                        this_numb = int(this_numb)
                        if this_numb < 0:
                            is_samp_b_zero = is_samp_b_zero + 1
                            is_error = is_error + 1
                        else:
                            thislist.append(elem[1])
                            thislist.append(this_numb)
                if selval == 1:
                    this_ov = strat_over_line.get("stratum" + str(elem[0]))
                    this_ov = this_ov.text()
                    if not this_ov:
                        is_oversamp = is_oversamp + 1
                        is_error = is_error + 1
                    else:
                        ovr_isint = self.integer_check(this_ov)
                        if ovr_isint == 1:
                            is_oversamp_int = is_oversamp_int + 1
                            is_error = is_error + 1
                        else:
                            this_ov = int(this_ov)
                            if this_ov < 0:
                                is_oversamp_b_zero = is_oversamp_b_zero + 1
                                is_error = is_error + 1
                            else:
                                thislist.append(this_ov)
                strat_list_to_append.append(thislist)
            if is_error != 0:
                error_text = []
                windw_er = QWidget()
                windw_er.setWindowTitle("Error window")
                label = QLabel("The following errors occured:")
                if is_samp != 0:
                    error_text.append("There are one or more sample numbers that are not given")
                if is_samp_int != 0:

                 error_text.append("There are one or more sample numbers that are not given in integer forms")
                if is_samp_b_zero != 0:
                    error_text.append("There are one or more sample numbers that are below zero")
                if selval == 1:
                    if is_oversamp != 0:
                        error_text.append("There are one or more oversample numbers that are not given")
                    if is_oversamp_int != 0:

                     error_text.append("There are one or more oversample numbers that are not given in integer form"
)
                    if is_oversamp_b_zero != 0:
                        error_text.append("There are one or more oversample numbers that are below zero")
                error_text = '\n'.join(error_text)
                textbox = QTextBrowser()
                textbox.append(error_text)
                ok_button = QPushButton("OK")
                ok_button.clicked.connect(lambda: self.strat_samp(windw_er, selval, layer, stratname_index))
                hbox = QHBoxLayout()
                hbox.addStretch(6)
                hbox.addWidget(ok_button)
                hbox.addStretch(6)
                vbox = QVBoxLayout()
                vbox.addWidget(label)
                vbox.addWidget(textbox)
                vbox.addLayout(hbox)
                windw_er.setLayout(vbox)
                windw_er.show()
            else:
                strat_info.append(strat_list_to_append)
                self.do_samp_strat(strat_info, layer, stratname_index)


        # Function for doing the sampling for the different strata:
        def do_samp_strat(self, strat_info, layer, stratname_index):
            usedlays = []
            fields = layer.fields()
            stratname_field = fields[stratname_index]
            stratname = stratname_field.name()
            origcrs = layer.crs().authid()
            if strat_info[0] == 2:
                for elem in strat_info[1]:
                    feats = layer.getFeatures()
                    sampl_lay = QgsVectorLayer("Polygon?crs={0}".format(origcrs), "temp_lay", "memory")
                    usedlays.append(sampl_lay)
                    pr = sampl_lay.dataProvider()
                    for f in fields:
                        pr.addAttributes([f])
                    sampl_lay.updateFields()
                    for feat in feats:
                        attrs = feat.attributes()
                        strat = attrs[stratname_index]
                        if elem[0] == strat:
                            pr.addFeatures([feat])
                    sampl_lay.updateExtents()
                    oversample = 0
                    is_oversampling = 1
                    is_windw = 0
                    is_from_strat = 1

                # Grid placement function called with default values for functionalities of the sampling from non s
tratified data:
```

```
1465
                self.gridplacement(sampl_lay, elem[1], 0, 1, oversample, is_oversampling, 0, is_windw, is_from_stra
t,
1466                                    elem[0])
1467                self.do_del_field(layer, stratname)
1468            if strat_info[0] == 1:
1469                for elem in strat_info[1]:
1470                    feats = layer.getFeatures()
1471                    sampl_lay = QgsVectorLayer("Polygon?crs={0}".format(origcrs), "temp_lay", "memory")
1472                    usedlays.append(sampl_lay)
1473                    pr = sampl_lay.dataProvider()
1474                    for f in fields:
1475                        pr.addAttributes([f])
1476                    sampl_lay.updateFields()
1477                    for feat in feats:
1478                        attrs = feat.attributes()
1479                        strat = attrs[stratname_index]
1480                        if elem[0] == strat:
1481                            pr.addFeatures([feat])
1482                    sampl_lay.updateExtents()
1483                    oversample = elem[2]
1484                    self.is_oversampling = 2
1485                    is_windw = 0
1486                    is_from_strat = 1
1487
                # Grid placement function called with default values for functionalities of the sampling from non s
tratified data:
1488                    self.gridplacement(sampl_lay, elem[1], 0, 1, oversample, 2, 0, is_windw, is_from_strat,
1489                                       elem[0])
1490                self.do_del_field(layer, stratname)
1491
1492
1493
1494
1495
1496        # ------------------------------------GRTS AREAL-----------------------------------------------------------
-------------
1497
1498
1499
1500
1501
1502        # Generator function to flatten multiple nested lists:
1503        def flatten(self, l):
1504            for el in l:
1505                if isinstance(el, collections.abc.Iterable) and not isinstance(el, (str, bytes)):
1506                    yield from self.flatten(el)
1507                else:
1508                    yield el
1509
1510
1511        # Error window for grid settings:
1512        def errorforquadr(self, error_text):
1513            windw = QWidget()
1514            windw.setWindowTitle("Error Window")
1515
        label = QLabel("The following error(s) occured (Please press OK to go back to the settings window):")
1516            text = QTextBrowser()
1517            error_text = ', '.join(error_text)
1518            text.append(error_text)
1519            ok_button = QPushButton()
1520            ok_button.setText("OK")
1521            ok_button.clicked.connect(lambda: self.error_ok(windw))
1522            vbox = QVBoxLayout()
1523            vbox.addWidget(label)
1524            vbox.addWidget(text)
1525            vbox.addStretch(3)
1526            vbox.addWidget(ok_button)
1527            windw.setLayout(vbox)
1528            windw.show()
1529
1530
1531        # Function for the error window if the while loop is longer than it should:
1532        def error_while(self):
1533            windw = QWidget()
1534            windw.setWindowTitle("ERROR WINDOW")
1535            label = QLabel(
1536
         "The polygon ID field generation lasted longer than it should! Please check the amount of fields with n
ame grts_id and a number!")
1537            cancel_button = QPushButton("CANCEL")
1538            cancel_button.clicked.connect(lambda: windw.close())
1539            vbox = QVBoxLayout()
1540            vbox.addWidget(label)
1541            vbox.addWidget(cancel_button)
1542            windw.setLayout(vbox)
1543            windw.show()
1544
1545
1546        # Function that closes the error window and opens the sampling settings window again:
1547        def error_ok(self, windw):
1548            windw.close()
1549            self.dlg.show()
1550            result = self.dlg.exec_()
1551            if result:
1552
            self.iscorrect(self.dlg.cbox.currentText(), self.dlg.numb.text(), self.dlg.custom_quadr.text(),
1553                           self.quadrchecker, self.dlg.oversample.text(), self.is_oversampling)
1554
1555
1556        # Checking if the number is an integer (returns 0 if it is an integer and 1 if it is not):
1557        def integer_check(self, n):
1558            notintvalue = 0
```

```
            try:
                float(n)
            except ValueError:
                notintvalue = notintvalue + 1
            if notintvalue == 0:
                if float(n) - round(float(n), 0) != 0:
                    notintvalue = notintvalue + 1
            if notintvalue != 0:
                isintvalue = 1
                return isintvalue
            else:
                isintvalue = 0
                return isintvalue

        # Function for using default quadrant ammount:
        def deftoggled(self, custom_quadr):
            self.quadrchecker = 1
            custom_quadr.setEnabled(False)

        # Function for using custom quadrant ammount:
        def customtoggled(self, custom_quadr):
            self.quadrchecker = 2
            custom_quadr.setEnabled(True)

        # Function for the disable oversampling button:
        def notoversample(self, oversample):
            self.is_oversampling = 1
            oversample.setEnabled(False)

        # Function for oversample button:
        def oversample(self, oversample):
            self.is_oversampling = 2
            oversample.setEnabled(True)

        # Function of the window asking back if the settings are correct:
        def iscorrect(self, layer, samp_number, custom, quadrchecker, oversample, is_oversampling):
            windw = QWidget()
            windw.setWindowTitle("Settings accepting")
            label = QLabel(
                "Are you sure that everything is correctly set?\nIf yes please press YES! If not please press NO!")
            no_button = QPushButton("NO")
            no_button.clicked.connect(lambda: self.error_ok(windw))
            yes_button = QPushButton("YES")

        # Calling the grid placement function with values that are set default because they are needed if the funct
ion is called from stratification:
            yes_button.clicked.connect(
                lambda: self.gridplacement(layer, samp_number, custom, quadrchecker, oversample, is_oversampling, windw
, 1,
                                           0, 0))
            vbox = QVBoxLayout()
            vbox.addWidget(label)
            hbox = QHBoxLayout()
            hbox.addWidget(yes_button)
            hbox.addWidget(no_button)
            vbox.addLayout(hbox)
            windw.setLayout(vbox)
            windw.show()

        # Grid placement on the layer:
        def gridplacement(self, layer, samp_number, custom, quadrchecker, oversample, is_oversampling, windw, is_windw,
                          is_from_strat, strat_name):
            if is_windw == 1:
                windw.close()
            erro_text = []
            is_error = 0
            samp_number_problem = 0
            oversample_porblem = 0
            if not layer:
                erro_text.append("A layer should be selected")
                is_error = is_error + 1
            if not samp_number:
                erro_text.append("A sample number must be given")
                is_error = is_error + 1
                samp_number_problem = samp_number_problem + 1
            else:
                isintvalue = self.integer_check(samp_number)
                if isintvalue != 0:
                    erro_text.append("The sample number must be an integer")
                    is_error = is_error + 1
                    samp_number_problem = samp_number_problem + 1
                else:
                    if int(samp_number) < 0:
                        erro_text.append("The number of desired samples must be over zero")
                        is_error = is_error + 1
                        samp_number_problem = samp_number_problem + 1
            if is_oversampling == 2:
                if not oversample:
                    erro_text.append("If oversampling is selected there should be a number of additional samples")
                    is_error = is_error + 1
                    oversample_porblem = oversample_porblem + 1
                else:
```

```
                    isintover = self.integer_check(oversample)
                    if isintover != 0:
                        erro_text.append("The number of additional samples must be an integer")
                        is_error = is_error + 1
                        oversample_porblem = oversample_porblem + 1
                    else:
                        if int(oversample) < 0:
                            erro_text.append("The number of additional samples must be over zero")
                            is_error = is_error + 1
                            oversample_porblem = oversample_porblem + 1
            if oversample_porblem == 0 and samp_number_problem == 0 and is_oversampling == 2:
                samp_number = int(samp_number) + int(oversample)
            if quadrchecker == 1:
                if samp_number_problem == 0 and oversample_porblem == 0:
                    cellgen = int(math.ceil(math.log(int(samp_number), 4) + 2))
                    cellnumb = 4 ** cellgen
            if quadrchecker == 2:
                if samp_number_problem == 0 and oversample_porblem == 0:
                    if not custom:
                        erro_text.append("The custom power of four must be given if custom is selected")
                        is_error = is_error + 1
                    else:
                        customisint = self.integer_check(custom)
                        if customisint != 0:
                            erro_text.append("The custom power of four must be an integer")
                            is_error = is_error + 1
                        else:
                            if int(custom) < 0:
                                erro_text.append("The custom power of four must be over zero")
                                is_error = is_error + 1
                            else:
                                cellgen = int(math.ceil(math.log(int(samp_number), 4) + 2))
                                if cellgen > int(custom):
                                    erro_text.append(

                                     "If custom quadrant number is selected, the selected power of 4 should be highe
r than the default(base 4 logarithm of the selected sample numbers plus 2)")
                                        is_error = is_error + 1
                                if int(custom) >= cellgen:
                                    cellgen = int(custom)
                                    cellnumb = 4 ** cellgen
            if is_error != 0:
                self.errorforquadr(erro_text)
            else:
                if is_from_strat == 0:
                    layer = QgsProject.instance().mapLayersByName(layer)[0]
                layextent = layer.extent()
                xextentmax = layextent.xMaximum()
                xextentmin = layextent.xMinimum()
                yextentmax = layextent.yMaximum()
                yextentmin = layextent.yMinimum()
                xextentmax = float(xextentmax)
                xextentmin = float(xextentmin)
                yextentmax = float(yextentmax)
                yextentmin = float(yextentmin)
                xrange = xextentmax - xextentmin
                yrange = yextentmax - yextentmin
                basenumb = int()
                if xrange > yrange:
                    basenumb = 1
                if yrange > xrange:
                    basenumb = 2
                if basenumb == 1:
                    rangedif = xrange - yrange
                    rangeadd = rangedif / 2
                    yextentmax = yextentmax + rangeadd
                    yextentmin = yextentmin - rangeadd
                    xextentmax = xextentmax
                    xextentmin = xextentmin
                if basenumb == 2:
                    rangedif = yrange - xrange
                    rangeadd = rangedif / 2
                    xextentmax = xextentmax + rangeadd
                    xextentmin = xextentmin - rangeadd
                    yextentmax = yextentmax
                    yextentmin = yextentmin
                cellx = (xextentmax - xextentmin) / (2 ** cellgen)
                celly = (yextentmax - yextentmin) / (2 ** cellgen)
                addx = (cellx * (2 ** cellgen)) / ((2 ** cellgen) - 1)
                addy = addx
                randfractx = round(uniform(1, 100), 1)
                randfracty = round(uniform(1, 100), 1)
                addx1 = addx / randfractx
                addx2 = addx - addx1
                addy1 = addy / randfracty
                addy2 = addy - addy1
                randnumbx = randint(1, 2)
                randnumby = randint(1, 2)
                if randnumbx == 1:
                    xextentmax = xextentmax + addx1
                    xextentmin = xextentmin - addx2
                if randnumbx == 2:
                    xextentmax = xextentmax + addx2
                    xextentmin = xextentmin - addx1
                if randnumby == 1:
                    yextentmax = yextentmax + addy1
                    yextentmin = yextentmin - addy2
                if randnumby == 2:
                    yextentmax = yextentmax + addy2
                    yextentmin = yextentmin - addy1
                cellx = (xextentmax - xextentmin) / (2 ** cellgen)
                celly = (yextentmax - yextentmin) / (2 ** cellgen)
```

```
1754
            extent = str(xextentmin) + ',' + str(xextentmax) + ',' + str(yextentmin) + ',' + str(yextentmax)
1755            crs = layer.crs()
1756            parameters = {'EXTENT': extent, 'HSPACING': cellx, 'VSPACING': celly, 'TYPE': 2, 'CRS': crs,
1757                          'OUTPUT': 'TEMPORARY_OUTPUT', 'HOVERLAY': 0, 'VOVERLAY': 0}
1758            grid = processing.run('qgis:creategrid', parameters)
1759            gridlay = grid['OUTPUT']
1760
            self.gridorder(cellgen, cellnumb, layer, gridlay, samp_number, oversample, is_oversampling, is_from_str
at,
1761                           strat_name)
1762
1763
1764        # Creating the order of cell IDs to match the here created hierarchical order of cell serial numbers:
1765
    def gridorder(self, cellgen, cellnumb, layer, gridlayer, samp_numb, oversample, is_oversampling, is_from_strat,

1766                  strat_name):
1767        numblist = []
1768        for i in range(1, (cellnumb + 1)):
1769            numblist.append(i)
1770        initlist = []
1771        for i in range(1, (2 ** cellgen + 1), 2):
1772            for x in range(1, (2 ** cellgen + 1), 2):
1773                number = numblist[i + ((x - 1) * (2 ** cellgen)) - 1]
1774                initlist.append(number)
1775        newlist = []
1776        for i in initlist:
1777
            listquadr = [numblist[i - 1], numblist[i + (2 ** cellgen) - 1], numblist[i], numblist[i + (2 ** cellgen
)]]
1778            newlist.append(listquadr)
1779        numblist = newlist
1780        if cellgen > 2:
1781            for i in range(1, (cellgen - 1)):
1782                usedlist = []
1783                newlist = []
1784                for x in range(len(numblist)):
1785                    if numblist[x] not in usedlist:
1786                        listquadr = [numblist[x], numblist[x + 1], numblist[x + (2 ** (cellgen - i))],
1787                                     numblist[x + (2 ** (cellgen - i)) + 1]]
1788                        newlist.append(listquadr)
1789                        usedlist.append(numblist[x])
1790                        usedlist.append(numblist[x + 1])
1791                        usedlist.append(numblist[x + (2 ** (cellgen - i))])
1792                        usedlist.append(numblist[x + (2 ** (cellgen - i)) + 1])
1793                numblist = newlist
1794        numblist = list(self.flatten(numblist))
1795        orderlist = [1, 2, 3, 4]
1796        shuffle(orderlist)
1797        for i in range(1, cellgen):
1798            newlist = list()
1799            for x in orderlist:
1800                randperm = [1, 2, 3, 4]
1801                shuffle(randperm)
1802                for y in randperm:
1803                    elem = str(x) + str(y)
1804                    newlist.append(elem)
1805            orderlist = newlist
1806        orderlist = list(map(int, orderlist))
1807        orderlist_old = copy.deepcopy(orderlist)
1808        if is_oversampling == 2:
1809            revord = []
1810            for elem in orderlist:
1811                elem = str(elem)
1812                elem = ''.join(reversed(elem))
1813                elem = int(elem)
1814                revord.append(elem)
1815            orderlist = revord
1816        gridfeatures = gridlayer.getFeatures()
1817        attrindex = gridlayer.fields().indexFromName("id")
1818        gridlayer.startEditing()
1819        for feat in gridfeatures:
1820            attrs = feat.attributes()
1821            idnumb = attrs[attrindex]
1822            indofid = numblist.index(idnumb)
1823            ordnumb = orderlist[indofid]
1824            gridlayer.changeAttributeValue(feat.id(), attrindex, ordnumb)
1825        gridlayer.commitChanges()
1826        self.sampl(gridlayer, layer, samp_numb, oversample, is_from_strat, strat_name)
1827
1828
1829    # Defining a function for grid sorting in the next section:
1830    def get_name(self, ft):
1831        return ft['id']
1832
1833
1834    # Function for creating the sample line:
1835    def sampl(self, gridlayer, layer, samp_numb, oversample, is_from_strat, strat_name):
1836
        # Creating an ID field (named grts_id) for polygons for future references (values will be assigned in the n
ext section):
1837        field_names = []
1838        for f in layer.fields():
1839            field_names.append(f.name())
1840        probenumb = 1
1841        id_error = 0
1842        timeout = time.time() + 60
1843        idfieldname_index = int()
1844        idfieldname_orig = str()
1845        while True:
1846            idfieldname = "grts_id" + str(probenumb)
1847            if idfieldname not in field_names:
```

```
                    layer.dataProvider().addAttributes([QgsField(idfieldname, QVariant.Int)])
                    layer.updateFields()
                    idfieldname_orig = copy.deepcopy(idfieldname)
                    idfieldname_index = layer.fields().indexFromName(idfieldname)
                    break
                if time.time() > timeout:
                    id_error = 1
                    break
                else:
                    probenumb = probenumb + 1
            if id_error == 1:
                self.error_while()
            else:
                idfieldname = idfieldname_orig
                # Getting whole polygons' inclusion probabilities
                # (by creating a list containing the list of ids, areas and line segments of each polygons)
                # (plus filling the polygon id field):
                getcontext().prec = 10
                layarea = Decimal()
                features = layer.getFeatures()
                for fet in features:
                    geom = fet.geometry()
                    area = geom.area()
                    area = Decimal(area)
                    layarea = layarea + area
                layarea = Decimal(layarea)
                wholeline = int(samp_numb) * 100000
                wholeline = Decimal(wholeline)
                polygon_probs = []
                idnumb = 1
                fts = layer.getFeatures()
                layer.startEditing()
                for feat in fts:
                    fid = feat.id()
                    layer.changeAttributeValue(fid, idfieldname_index, idnumb)
                    polylist = [idnumb]
                    geom = feat.geometry()
                    area = geom.area()
                    area = Decimal(area)
                    polylist.append(area)
                    area_ratio = area / layarea
                    polyline = wholeline * area_ratio
                    polyline = int(polyline)
                    polylist.append(polyline)
                    polygon_probs.append(polylist)
                    idnumb = idnumb + 1
                layer.updateFields()
                layer.commitChanges()
                # Cutting the grid layer by the original layer to be sampled:
                params = {'INPUT': gridlayer, 'OVERLAY': layer, 'OUTPUT': 'TEMPORARY_OUTPUT'}
                cutted = processing.run('native:clip', params)
                cuttedlayer = cutted['OUTPUT']
                # Sorting grid features by the id:
                feats = cuttedlayer.getFeatures()
                idindex = cuttedlayer.fields().indexFromName("id")
                feats = sorted(feats, key=self.get_name)
                # Looping through the sorted features and get the data of its line segment

            # (a list of two lists; in the first list there is the quadrant id and will be the extent of the line s
egment,

            # and in the second list there are lists containing the id of the polygon and the extent of it inside t
he quadrant):
                sample_order_list = []
                line_length = 0
                begin = 0
                begin_for_whole_quadrant = 0
                thisdict = {}
                for feat in feats:

                    # First assign the id of the grid to the first list and then get the polygons inside the grid:
                    featattrs = feat.attributes()
                    feat_order = featattrs[idindex]
                    ordrange = [feat_order]
                    fid = feat.id()
                    feataslay = cuttedlayer.materialize(QgsFeatureRequest().setFilterFid(fid))
                    pars = {'INPUT': layer, 'OVERLAY': feataslay, 'OUTPUT': 'TEMPORARY_OUTPUT'}
                    onelay = processing.run('native:clip', pars)
                    onelayer = onelay['OUTPUT']
                    thisdict["grid_" + str(feat_order)] = onelayer
                    # Creating the information list of the polygons (id, order, length in the line):
                    ft = onelayer.getFeatures()
                    polylist = []
                    thisord = 1
                    for fit in ft:
                        featdata = []
                        featattrs = fit.attributes()
                        featid = featattrs[idfieldname_index]
                        featdata.append(featid)
                        featdata.append(thisord)
                        for elem in polygon_probs:
                            if int(elem[0]) == int(featid):
                                geom = fit.geometry()
                                area = geom.area()
                                area = Decimal(area)
                                arearatio = area / elem[1]
                                arearatio = Decimal(arearatio)
                                featlength = elem[2] * arearatio
                                featlength = int(featlength)
                                featdata.append(featlength)
                            else:
                                pass
                        polylist.append(featdata)
```

```
                        thisord = thisord + 1

                # Creating the random permutation of the polygons and its place on the line (It is the final_order_
polygons)
                    # and calculating the length of the whole line of the quadrant:
                    randperm = [*range(1, (thisord + 1))]
                    shuffle(randperm)
                    final_order_polygons = []
                    whole_line_length = int()
                    for i in randperm:
                        for el in polylist:
                            if el[1] == i:
                                polydata = [el[0]]
                                polyplace = [*range(begin, (begin + el[2] + 1))]
                                polydata.append(polyplace)
                                final_order_polygons.append(polydata)
                                whole_line_length = whole_line_length + el[2]
                                begin = begin + el[2]
                            else:
                                pass
                    # Assigning the whole line length to the list containing the polygon id

                # and assigning the quadrants line length to the variable holding the whole line (line_length):
                    line_length = line_length + whole_line_length
                    whole_line_length_as_number = copy.deepcopy(whole_line_length)
                    whole_line_length = [
                        *range(begin_for_whole_quadrant, (begin_for_whole_quadrant + whole_line_length + 1))]
                    ordrange.append(whole_line_length)
                    begin_for_whole_quadrant = begin_for_whole_quadrant + whole_line_length_as_number
                    # Creating a list of the quadrant infromation list (ordrange)
                    # and the polygon information list (final_order_polygons):
                    final_list = [ordrange, final_order_polygons]
                    # Assigning this final list to the final list of the quadrant line information:
                    sample_order_list.append(final_list)
                # Calling the next function to sample the line created in here:

            self.getsample(sample_order_list, line_length, samp_numb, thisdict, layer, cuttedlayer, idfieldname, gr
idlayer,
                           oversample, is_from_strat, strat_name)


        # Function for get the element for samples sorting in the next section:
        def sampsort(self, sample):
            return sample[0]


        # Function for getting the sample:

    def getsample(self, sample_order_list, line_length, samp_numb, thisdict, layer, cuttedlayer, idfieldname, gridl
ayer,
                  oversample, is_from_strat, strat_name):
            # Getting the line points from a systematic sampling method:
            recent_elem = randint(1, line_length)
            line_numbers = [recent_elem]
            line_step = line_length / int(samp_numb)
            for y in range(1, int(samp_numb)):
                line_numb = recent_elem + line_step
                if line_numb <= line_length:
                    line_numb = math.trunc(line_numb)
                    line_numbers.append(line_numb)
                    recent_elem = line_numb
                else:
                    line_numb = line_numb - line_length
                    line_numb = math.trunc(line_numb)
                    line_numbers.append(line_numb)
                    recent_elem = line_numb
            # Getting the ids from the previously got line points:
            samples = []
            for elem in sample_order_list:
                for samp in line_numbers:
                    if samp in elem[0][1]:
                        for el in elem[1]:
                            if samp in el[1]:
                                this_samp = [elem[0][0], el[0]]
                                samples.append(this_samp)

        # Creating a layer for the sample points (with 4 attributes, order_number, serial_number, x_coordinate, y_c
oordinate):
            origcrs = layer.crs().authid()
            if is_from_strat == 0:

            sample_point_layer = QgsVectorLayer("Point?crs={0}".format(origcrs), "sample_points", "memory")
            if is_from_strat == 1:
                namelay = "sample_points" + str(strat_name)
                sample_point_layer = QgsVectorLayer("Point?crs={0}".format(origcrs), namelay, "memory")
            pr = sample_point_layer.dataProvider()
            sample_point_layer.startEditing()

        pr.addAttributes([QgsField("Order_number", QVariant.String), QgsField("Serial_number", QVariant.String),

                          QgsField("X_Coordinate", QVariant.Double), QgsField("Y_Coordinate", QVariant.Double)])
            sample_point_layer.updateFields()
            QgsProject.instance().addMapLayer(sample_point_layer)
            # Creating a layer for the oversample points if oversampling is selected:
            if self.is_oversampling == 2:
                if is_from_strat == 0:

                oversample_point_layer = QgsVectorLayer("Point?crs={0}".format(origcrs), "oversample_points", "memo
ry")
                if is_from_strat == 1:
                    nameovlay = "oversample_points" + str(strat_name)

                oversample_point_layer = QgsVectorLayer("Point?crs={0}".format(origcrs), nameovlay, "memory")
```

```
                opr = oversample_point_layer.dataProvider()
                oversample_point_layer.startEditing()

            opr.addAttributes([QgsField("Order_number", QVariant.String), QgsField("Serial_number", QVariant.String
),
                               QgsField("X_Coordinate", QVariant.Double), QgsField("Y_Coordinate", QVariant.Double)
])
                oversample_point_layer.updateFields()
                QgsProject.instance().addMapLayer(oversample_point_layer)
            # Taking-
apart the oversample component of the sample number from the real sample amount (if there is oversampling):
            if self.is_oversampling == 2:
                samp_numb = int(samp_numb) - int(oversample)
            # Generating random points in the selected polygon fragments:
            cutfeats = cuttedlayer.getFeatures()
            cutfeats = sorted(cutfeats, key=self.get_name)
            idindex = cuttedlayer.fields().indexFromName("id")
            samples = sorted(samples, key=self.sampsort)
            serial = 1
            samp_counter = 0
            for cf in cutfeats:
                if samp_counter == samp_numb:
                    break
                cfatts = cf.attributes()
                for sampl in samples:
                    if cfatts[idindex] == sampl[0]:
                        samplay = thisdict.get("grid_" + str(cfatts[idindex]))
                        samppointlayfeats = samplay.getFeatures()
                        smplf_id_index = samplay.fields().indexFromName(idfieldname)
                        for smplf in samppointlayfeats:
                            if smplf[smplf_id_index] == sampl[1]:
                                fid = smplf.id()
                                laytorandpoint = samplay.materialize(QgsFeatureRequest().setFilterFid(fid))
                                parameters = {'INPUT': laytorandpoint, 'MIN_DISTANCE': 0, 'POINTS_NUMBER': 1,
                                              'OUTPUT': 'TEMPORARY_OUTPUT'}
                                rand = processing.run('qgis:randompointsinlayerbounds', parameters)
                                randlayer = rand['OUTPUT']
                                rand_features = randlayer.getFeatures()
                                for randfet in rand_features:
                                    randgeom = randfet.geometry()
                                fet = QgsFeature()
                                fet.setGeometry(randgeom)

                                fet.setAttributes([str(cfatts[idindex]), str(serial), str(randgeom.asPoint()[0]),
                                                   str(randgeom.asPoint()[1])])
                                pr.addFeatures([fet])
                                serial = serial + 1
                                sampind = samples.index(sampl)
                                del samples[sampind]
                                samp_counter = samp_counter + 1
                                break
                            else:
                                pass
            # Additional sample creation if oversampling is enabled:
            if self.is_oversampling == 2:
                oscutfeats = cuttedlayer.getFeatures()
                oscutfeats = sorted(cutfeats, key=self.get_name)
                serial = 1
                for oscf in oscutfeats:
                    oscfatts = oscf.attributes()
                    for samp in samples:
                        if oscfatts[idindex] == samp[0]:
                            samplayer = thisdict.get("grid_" + str(oscfatts[idindex]))
                            samppointlayerfeats = samplayer.getFeatures()
                            smplrf_id_index = samplayer.fields().indexFromName(idfieldname)
                            for smplrf in samppointlayerfeats:
                                if smplrf[smplrf_id_index] == samp[1]:
                                    fd = smplrf.id()

                                layertorandpoint = samplayer.materialize(QgsFeatureRequest().setFilterFid(fd))

                                params = {'INPUT': layertorandpoint, 'MIN_DISTANCE': 0, 'POINTS_NUMBER': 1,
                                          'OUTPUT': 'TEMPORARY_OUTPUT'}
                                osrand = processing.run('qgis:randompointsinlayerbounds', params)
                                osrandlayer = osrand['OUTPUT']
                                osrand_features = osrandlayer.getFeatures()
                                for osrandfet in osrand_features:
                                    osrandgeom = osrandfet.geometry()
                                osfet = QgsFeature()
                                osfet.setGeometry(osrandgeom)

                                osfet.setAttributes([str(oscfatts[idindex]), str(serial), str(osrandgeom.asPoint()[
0]),
                                                     str(osrandgeom.asPoint()[1])])
                                opr.addFeatures([osfet])
                                serial = serial + 1
                                sampind = samples.index(samp)
                                del samples[sampind]
                                break
                            else:
                                pass
            ind = layer.fields().indexFromName(str(idfieldname))
            dellist = [ind]
            prov = layer.dataProvider()
            prov.deleteAttributes(dellist)
            layer.updateFields()
```

```
2126        # ---------------------------- DELETING FIELD STORING THE STRATUM NAMES ---------------------------------
------------------------------
2127
2128
2129
2130
2131
2132
2133        # Function for the window asking if deleting or keeping the field is preferred:
2134        def do_del_field(self, layer, stratname):
2135            windw = QWidget()
2136            windw.setWindowTitle("Stratum name field deletion")
2137            label = QLabel("Do you want to keep the field storing tha stratum name for each feature?")
2138            yes_button = QPushButton("YES")
2139            yes_button.clicked.connect(lambda: windw.close())
2140            no_button = QPushButton("NO")
2141            no_button.clicked.connect(lambda: self.delfield(layer, stratname, windw))
2142            hbox = QHBoxLayout()
2143            hbox.addWidget(yes_button)
2144            hbox.addWidget(no_button)
2145            vbox = QVBoxLayout()
2146            vbox.addWidget(label)
2147            vbox.addLayout(hbox)
2148            windw.setLayout(vbox)
2149            windw.show()
2150
2151
2152        # Function for deleting the field if no is selected:
2153        def delfield(self, layer, stratname, windw):
2154            windw.close()
2155            ind = layer.fields().indexFromName(str(stratname))
2156            dellist = [ind]
2157            prov = layer.dataProvider()
2158            prov.deleteAttributes(dellist)
2159            layer.updateFields()
2160
2161
2162
2163
2164
2165        # ---------------------------- CREATING THE DIALOG WINDOW ----------------------------------------------
---------------
2166
2167
2168
2169
2170
2171        def run(self):
2172            """Run method that performs all the real work"""
2173            # Create the dialog with elements (after translation) and keep reference
2174            # Only create GUI ONCE in callback, so that it will only load when the plugin is started
2175            if self.first_start == True:
2176                self.first_start = False
2177                self.dlg = VetSampDialog()
2178            # Get the loaded layers in the combobox:
2179            self.dlg.cbox.clear()
2180            loaded_layers = self.iface.mapCanvas().layers()
2181            vector_layers = []
2182            for layer in loaded_layers:
2183                if layer.type() == QgsMapLayer.VectorLayer:
2184                    vector_layers.append(layer)
2185            loaded_layers = vector_layers
2186            loaded_layers_name = []
2187            for layer in loaded_layers:
2188                name = layer.name()
2189                loaded_layers_name.append(name)
2190            self.dlg.cbox.addItems(loaded_layers_name)
2191            # Set the radiobuttons and linedit for custom quadrant number:
2192            self.quadrchecker = 1
2193            self.dlg.custom_quadr.setEnabled(False)
2194            grid_group = QButtonGroup(self.dlg)
2195            self.dlg.button_def.setChecked(True)
2196            self.dlg.button_def.toggled.connect(lambda: self.deftoggled(self.dlg.custom_quadr))
2197            self.dlg.button_custom.toggled.connect(lambda: self.customtoggled(self.dlg.custom_quadr))
2198            grid_group.addButton(self.dlg.button_def)
2199            grid_group.addButton(self.dlg.button_custom)
2200            # Set the radiobuttons and lineedit for oversampling:
2201            self.is_oversampling = 1
2202            self.dlg.oversample.setEnabled(False)
2203            oversampling_group = QButtonGroup(self.dlg)
2204            self.dlg.button_not_oversampling.setChecked(True)
2205            self.dlg.button_not_oversampling.toggled.connect(lambda: self.notoversample(self.dlg.oversample))
2206            self.dlg.button_oversampling.toggled.connect(lambda: self.oversample(self.dlg.oversample))
2207            oversampling_group.addButton(self.dlg.button_not_oversampling)
2208            oversampling_group.addButton(self.dlg.button_oversampling)
2209            # Setting the checkbox for enabling stratified sampling:
2210            self.dlg.strat_button.clicked.connect(lambda: self.is_strat_sel())
2211            # Setting the name of the dialog:
2212            self.dlg.setWindowTitle("VetSamp sampling settings")
2213            # show the dialog
2214            self.dlg.show()
2215            # Run the dialog event loop
2216            result = self.dlg.exec_()
2217            # See if OK was pressed
2218            if result:
2219
             self.iscorrect(self.dlg.cbox.currentText(), self.dlg.numb.text(), self.dlg.custom_quadr.text(),
2220                           self.quadrchecker, self.dlg.oversample.text(), self.is_oversampling)
```

Konzulensi ellenjegyzés

Alulírott Dr. Solymosi Norbert igazolom, hogy Papp Márton János *Földrajzi mintavételezési QGIS-plugin fejlesztése* című diplomamunkáját ismerem, azt beadásra és védésre alkalmasnak tartom.

Budapest, 2019.11.22.

.....

Dr. Solymosi Norbert

Bioinformatikai Központ

Nyilatkozat a TDK és a diplomamunka azonosságáról

Alulírott Papp Márton János nyilatkozom, hogy diplomamunkám, melynek címe *Földrajzi mintavételezési QGIS-plugin fejlesztése* tartalmi és formai szempontból teljes mértékben megegyezik az azonos című, a 2019.évi TDK konferencián szerepelt dolgozatommal.

Budapest, 2019.11.22.

.....

Papp Márton János

HuVetA

ELHELYEZÉSI MEGÁLLAPODÁS ÉS SZERZŐI JOGI NYILATKOZAT*

Név: Papp Márton János
Elérhetőség (e-mail cím): pappmarci95@gmail.com
A feltöltendő mű címe: Földrajzi mintavételezési QGIS-plugin fejlesztése
A mű megjelenési adatai: Diplomamunka
Az átadott fájlok száma: 1

Jelen megállapodás elfogadásával a szerző, illetve a szerzői jogok tulajdonosa nem kizárólagos jogot biztosít a HuVetA számára, hogy archiválja (a tartalom megváltoztatása nélkül, a megőrzés és a hozzáférhetőség biztosításának érdekében) és másolásvédett PDF formára konvertálja és szolgáltassa a fenti dokumentumot (beleértve annak kivonatát is).

Beleegyezik, hogy a HuVetA egynél több (csak a HuVetA adminisztrátorai számára hozzáférhető) másolatot tároljon az Ön által átadott dokumentumból kizárólag biztonsági, visszaállítási és megőrzési célból.

Kijelenti, hogy az átadott dokumentum az Ön műve, és/vagy jogosult biztosítani a megállapodásban foglalt rendelkezéseket arra vonatkozóan. Kijelenti továbbá, hogy a mű eredeti és legjobb tudomása szerint nem sérti vele senki más szerzői jogát. Amennyiben a mű tartalmaz olyan anyagot, melyre nézve nem Ön birtokolja a szerzői jogokat, fel kell tüntetnie, hogy korlátlan engedélyt kapott a szerzői jog tulajdonosától arra, hogy engedélyezhesse a jelen megállapodásban szereplő jogokat, és a harmadik személy által birtokolt anyagrész mellett egyértelműen fel van tüntetve az eredeti szerző neve a művön belül.

A szerzői jogok tulajdonosa a hozzáférés körét az alábbiakban határozza meg (**egyetlen, a megfelelő négyzetben elhelyezett x jellel**):

[X] engedélyezi, hogy a HuVetA-ban -ban tárolt művek korlátlanul hozzáférhetővé váljanak a világhálón,

[ ] az Állatorvostudományi Egyetem belső hálózatára (IP címeire) korlátozza a feltöltött dokumentum(ok) elérését,

[ ] a Könyvtárban található, dedikált elérést biztosító számítógépre korlátozza a feltöltött dokumentum(ok) elérését,

[ ] csak a dokumentum bibliográfiai adatainak és tartalmi kivonatának feltöltéséhez járul hozzá (korlátlan hozzáféréssel),

Kérjük, **nyilatkozzon a négyzetben elhelyezett jellel a helyben használatról is:**

☒ Engedélyezem a dokumentum(ok) nyomtatott változatának helyben olvasását a könyvtárban.

Amennyiben a feltöltés alapját olyan mű képezi, melyet valamely cég vagy szervezet támogatott illetve szponzorált, kijelenti, hogy jogosult egyetérteni jelen megállapodással a műre vonatkozóan.

A HuVetA üzemeltetői a szerző, illetve a jogokat gyakorló személyek és szervezetek irányában nem vállalnak semmilyen felelősséget annak jogi orvoslására, ha valamely felhasználó a HuVetÁ-ban engedéllyel elhelyezett anyaggal törvénysértő módon visszaélne.

Budapest, 2015. év ....11.....hó ..22...nap

aláírás
szerző/a szerzői jog tulajdonosa

---

*A **HuVetAMagyar Állatorvos-tudományi Archívum – Hungarian Veterinary Archive** az Állatorvostudományi Egyetem Hutyra Ferenc Könyvtár, Levéltár és Múzeum által működtetett egyetemi és szakterületi online adattár, melynek célja, hogy a magyar állatorvos-tudomány és -történet dokumentumait, tudásvagyonát elektronikus formában összegyűjtse, rendszerezze, megőrizze, kereshetővé és hozzáférhetővé tegye, szolgáltassa, a hatályos jogi szabályozások figyelembe vételével.*

A HuVetA a korszerű informatikai lehetőségek felhasználásával biztosítja a könnyű, (internetes keresőgépekkel is működő) kereshetőséget és lehetőség szerint a teljes szöveg azonnali elérését. Célja ezek révén

- *a magyar állatorvos-tudomány hazai és nemzetközi ismertségének növelése;*
- *a magyar állatorvosok publikációira történő hivatkozások számának, és ezen keresztül a hazai állatorvosi folyóiratok impakt faktorának növelése;*
- *az Állatorvostudományi Egyetem és az együttműködő partnerek tudásvagyonának koncentrált megjelenítése révén az intézmények és a hazai állatorvos-tudomány tekintélyének és versenyképességének növelése;*
- *a szakmai kapcsolatok és együttműködés elősegítése,*
- *a nyílt hozzáférés támogatása.*