Geostatisztika alapjai

Térinformatika R-ben

2023.11.27.

Section 1

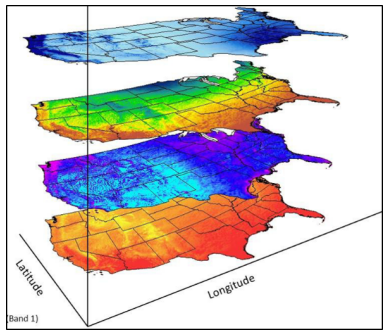
Haladó raszteralgebra

Haladó raszteralgebra

Raszteralgebra

- raszterekkel számos műveletet (pl. összeadás) elvégezhetünk kényelmesen
- akár egy vagy több raszter is részt vehet a műveletben
- több raszter esetén a dimenziók, felbontás stb. egyezzenek meg! (cellánkénti művelet)
- az eredmény mindig egy raszter, hasonló dimenziókkal, felbontással stb.

Haladó raszteralgebra



Raszteralgebra függvényei

- alapvető műveletek: `+`, `-`, `*`, `/` stb.
- alapvető statisztikai függvények: `min(...)`, `max(...)`, `sum(...)`, `mean(x)` stb.
- és még nagyon sok más művelet és függvény
- a függvények és műveletek természetesen halmozhatóak is
- ha ez nem lenne elég: `calc()` és `overlay()`

Haladó raszteralgebra

```
library(elsa)
library(raster)
library(rasterVis)
library(ncf)
library(sf)
domborzatmodell <- raster("domborzatmodell.tif")
csapadek <- raster("csapadek.tif")
felszinboritas <- raster("felszinboritas.tif")
felszinhomerseklet <- raster("felszinhomerseklet.tif")
ortofoto <- brick("ortofoto.tif")
load("eghajlat.RData")
```

Haladó raszteralgebra

Létrehozunk néhány azonos cellaszámú/felbontású rasztert:

```
esztergom_dem <- domborzatmodell[1:150, 1:300, drop = FALSE]
esztergom_homerseklet <- crop(x = felszinhomerseklet, y =
 esztergom_dem)
esztergom_felszinboritas <- projectRaster(from =
 felszinboritas, to = esztergom_dem)
```

Haladó raszteralgebra

```
summary(esztergom_dem[])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   99.0   148.0   191.0   203.3   239.0   588.0    9294
```

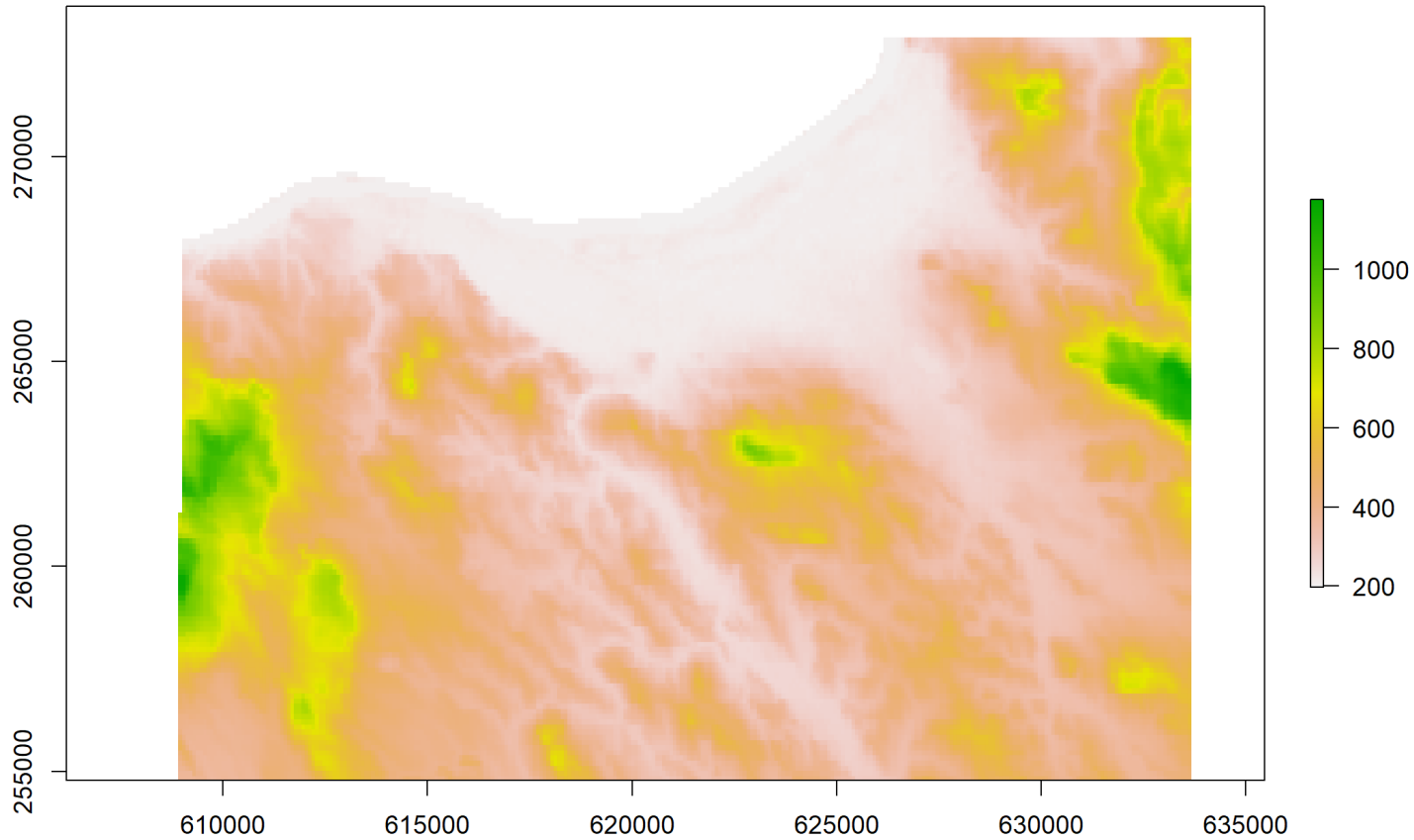
```
magas_hegyek <- esztergom_dem * 2
```

```
summary(magas_hegyek[])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  198.0   296.0   382.0   406.6   478.0  1176.0    9294
```

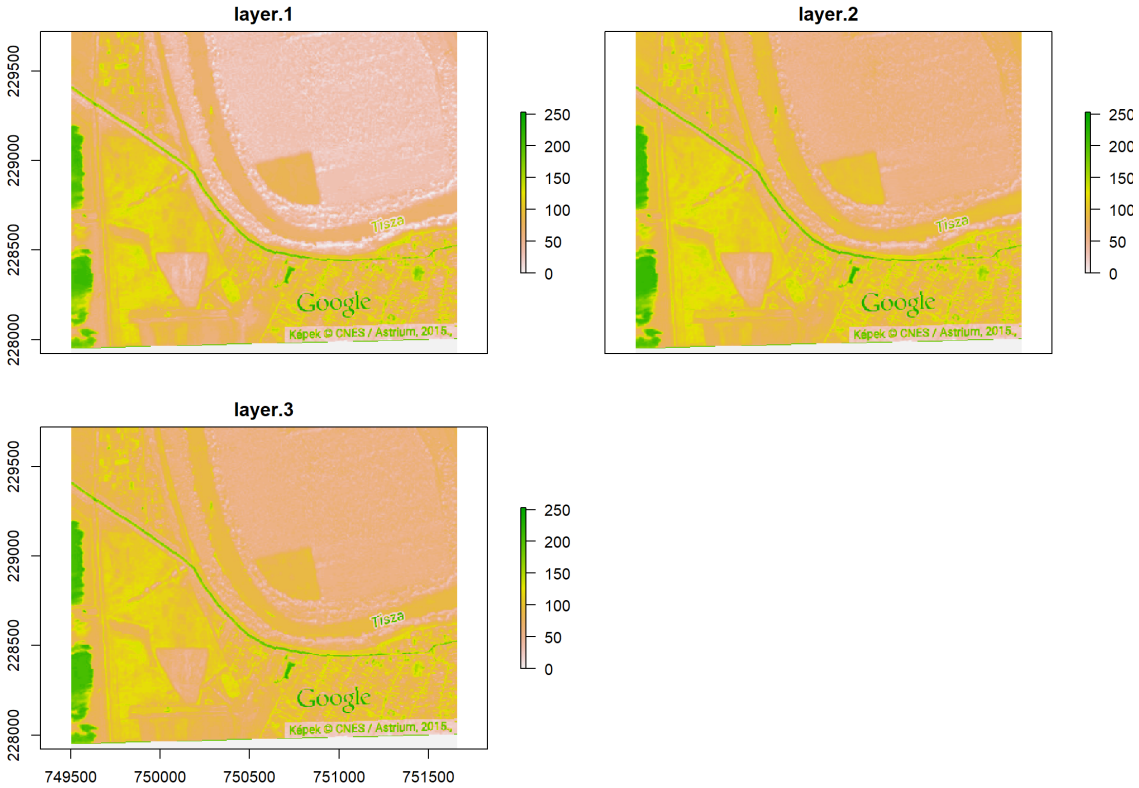
Haladó raszteralgebra

```
plot(magas_hegyek)
```



Haladó raszteralgebra

```
tisza_kanyar <- ortofoto[800:1300, 900:1500, drop = FALSE]
plot(tisza_kanyar)
```



Haladó raszteralgebra

Több raszterrel is dolgozhatunk:

```
names(tisza_kanyar) <- c("piros", "zold", "kek")
vilagossag1 <- (tisza_kanyar$piros + tisza_kanyar$zold +
 tisza_kanyar$kek) / 3
plot(vilagossag1, col = gray.colors(10))
```

Haladó raszteralgebra

Példa statisztikai függvényre:

```
vilagossag2 <- mean(tisza_kanyar)
plot(vilagossag2, col = gray.colors(10))
```

Ha nem a cellákat akarjuk a rétegek között átlagolni, hanem az egyes rétegekre külön számolnánk statisztikát: `cellStats()` (később...)

Haladó raszteralgebra

Függvényeket és műveleteket tetszőlegesen kombinálhatunk.

```
valami_bonyolult <- sqrt(floor(sum(tisza_kanyar) / 5))
plot(valami_bonyolult, col = gray.colors(10))
```

Haladó raszteralgebra

`calc()` és `overlay()`

- nem minden függvényt és műveletet tudunk a raszterekkel használni
- illetve nagy raszterek esetén óriási ideiglenes raszterek képződnének
- ilyenkor használható a `calc()` és `overlay()`
- mindkettő nagyon rugalmas, gyakorlatilag tetszőlegesen összetett műveletre rábírhatjuk őket (saját függvénnyel)
- a `calc()` az egyszerűbb
- most mindkettőt nagyon egyszerű példákon keresztül ismerjük meg (nem írunk saját függvényt)
- ezért a funkcionalitásukra/rugalmasságukra csak ímmel-ámmal derül majd fény

Haladó raszteralgebra a `calc()` függvénnyel

`calc(x, fun, na.rm)`

- egy többrétegű raszter minden cellájára a rétegekben található értékek alapján új rétege(ke)t képez
- `x`: a bemeneti többrétegű raszter
- `fun`: az alkalmazandó függvény, amely egy számvektort vár bemenetként
- a számvektornak annyi eleme lesz, ahány rétege `x`-nek
- a függvény visszaadhat akár egy, akár több értéket
- na.rm: elhagyjuk-e az ismeretlen értékeket (a legtöbb függvény esetén alapértelmezetten: nem)
- eredmény: egyrétegű vagy többrétegű raszter
- a rétegek számát a függvény visszatérési értékének hossza határozza meg

Haladó raszteralgebra a `calc()` függvénnyel

Harmadik variáció az átlagképzésre:

```
vilagossag3 <- calc(x = tisza_kanyar, fun = mean)
plot(vilagossag3, col = gray.colors(10))
```

Haladó raszteralgebra a `calc()` függvénnyel

A `range()` függvény a bemeneti vektor legkisebb és legnagyobb értékét adja vissza, vagyis egy kételemű számvektort.
Az eredmény ennek megfelelően többrétegű lesz, ha a `calc()`-kal használjuk.

```
range(c(3, 5, 2, 7))
```

```
[1] 2 7
```

```
tartomany <- calc(x = tisza_kanyar, fun = range)
class(tartomany)
```

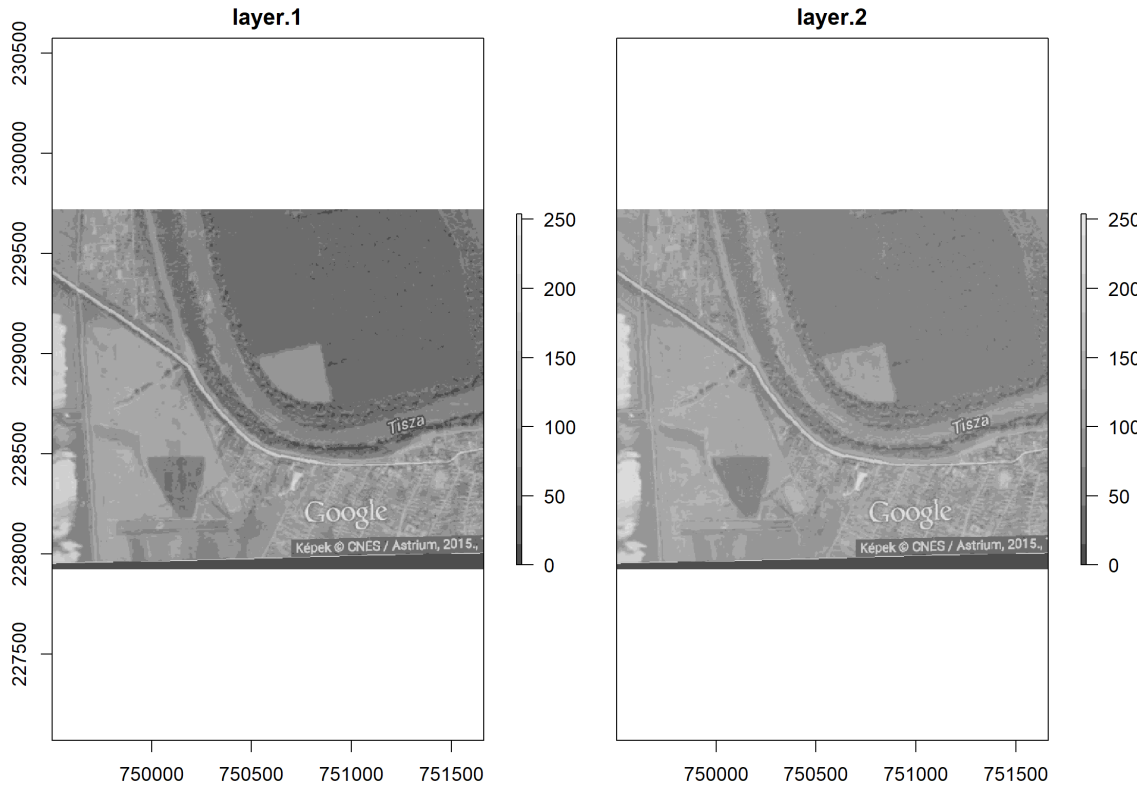
```
[1] "RasterBrick"
attr(,"package")
[1] "raster"
```

```
nlayers(tartomany)
```

```
[1] 2
```

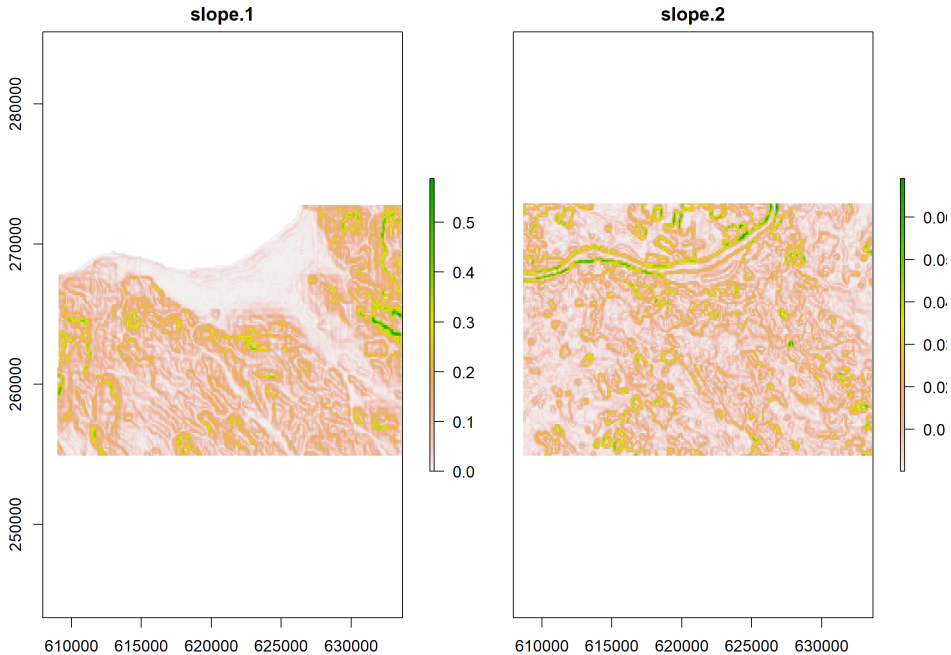
Haladó raszteralgebra a `calc()` függvénnyel

```
plot(tartomany, col = gray.colors(10))
```



Haladó raszteralgebra a `calc()` függvénnyel

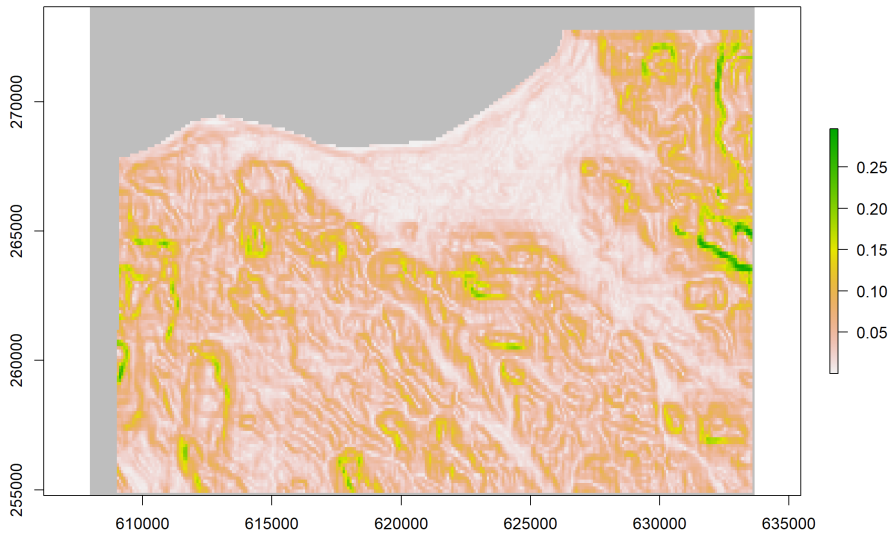
```
lejto <- terrain(x = esztergom_dem, opt = "slope")
homersekletvaltozas_merteke <- terrain(x =
 esztergom_homerseklet, opt = "slope")
valtozasok <- stack(lejto, homersekletvaltozas_merteke)
plot(valtozasok)
```



Haladó raszteralgebra a `calc()` függvénnyel

Átlagoljuk az almát a körtével!

```
atlagos_valtozas <- calc(x = valtozasok, fun = mean)
plot(atlagos_valtozas, colNA = "gray")
```

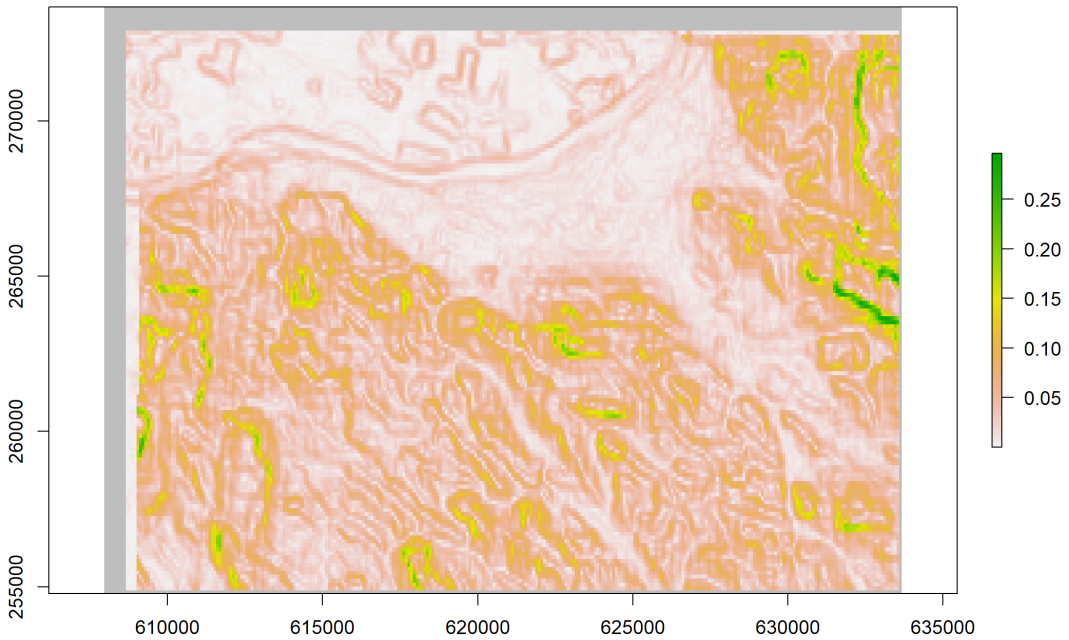


Az `na.rm` paraméter alapesetben `FALSE`, ezért a domborzatmodellben nem szereplő, határon túli helyek ismeretlenek.

Haladó raszteralgebra a `calc()` függvénnyel

Ugyanez `na.rm = TRUE` beállítással:

```
atlagos_valtozas <- calc(x = valtozasok, fun = mean, na.rm
 = TRUE)
plot(atlagos_valtozas, colNA = "gray")
```



Haladó raszteralgebra az `overlay()` függvénnyel

`overlay(x, y, …, fun)`

- lehet használni a `calc()`-hoz hasonló módon is, egy darab többrétegű rasztert (`x`) adva neki
- mi ezt most nem nézzük meg, mert erre tökéletes a `calc()`…
- értelmesebb eset: több darab (egy- vagy) többrétegű rasztert adunk neki (`x`, `y`, `...`)
- `fun`: egy olyan függvény, ami pont annyi paramétert (vagy tetszőleges számút (`...`)) vár, ahány bemeneti raszterünk van
- fontos, hogy a függvény vektorokat is kezelni tudjon!
- a függvényt először mindegyik bemeneti raszter 1. rétegére futtatja, majd a 2. rétegekre stb.
- eredmény: egy annyi rétegű raszter, amennyi rétege a bemeneti raszteréknek volt (vagy annak többszöröse)

Haladó raszteralgebra az `overlay()` függvénnyel

Egyszerű eset (egyrétegű raszterek):

```
osszeg <- overlay(x = esztergom_dem, y =
 esztergom_homerseklet, esztergom_felszinboritas, fun =
 sum)
```

Ilyenkor sok értelme nincs, hogy `overlay()`-t használunk.
Nyugodtan lehetne összefűzni a rétegeket, és a calc()-ot használni, eképpen:

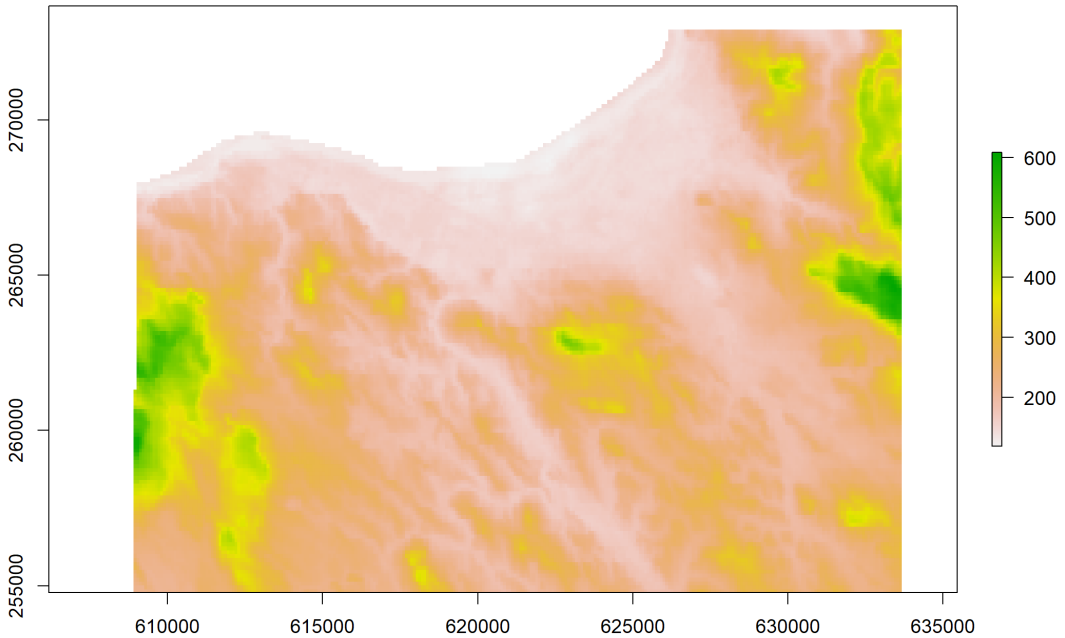
```
osszeg <- calc(x = stack(esztergom_dem,
 esztergom_homerseklet, esztergom_felszinboritas), fun =
 sum)
```

FONTOS: ez a megállapítás csak kellően rugalmas függvények (pl. `sum()`, `min()`, `max()`) esetében igaz!

Haladó raszteralgebra az `overlay()` függvénnyel

Az eredmény egyrétegű, hiszen a bemeneti raszterek is egyrétegűek voltak, és az alkalmazott függvény egy számot ad eredményül.

```
plot(osszeg)
```



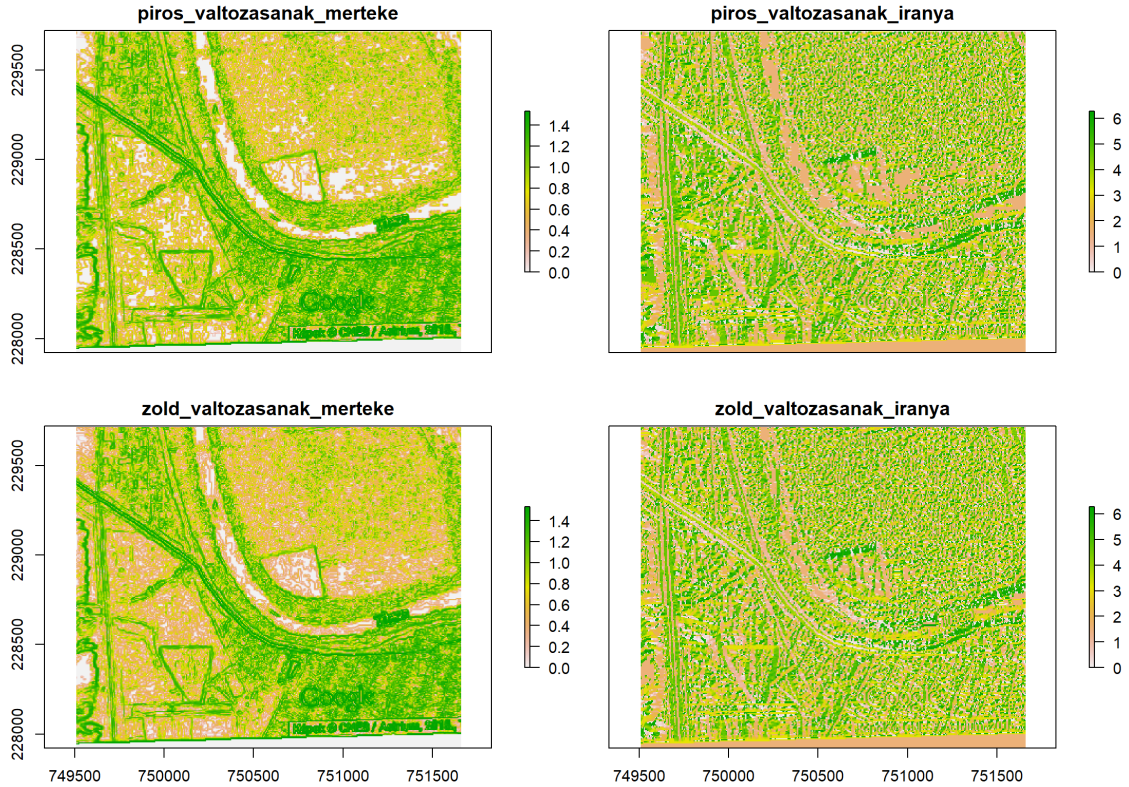
Haladó raszteralgebra az `overlay()` függvénnyel

Létrehozunk többrétegű rasztereket.

```
piros_valtozas <- stack(terrain(x = tisza_kanyar$piros,
 opt = "slope"), terrain(x = tisza_kanyar$piros, opt =
 "aspect"))
names(piros_valtozas) <- c("piros_valtozasanak_merteke",
 "piros_valtozasanak_iranya")
zold_valtozas <- stack(terrain(x = tisza_kanyar$zold, opt
 = "slope"), terrain(x = tisza_kanyar$zold, opt =
 "aspect"))
names(zold_valtozas) <- c("zold_valtozasanak_merteke",
 "zold_valtozasanak_iranya")
```

Haladó raszteralgebra az `overlay()` függvénnyel

```
plot(stack(piros_valtozas, zold_valtozas))
```



Haladó raszteralgebra az overlay() függvénnyel

Az overlay() használata több többrétegű raszterrel:

```
valtozasok_atlaga <- overlay(x = piros_valtozas, y =
 zold_valtozas, fun = mean)
nlayers(valtozasok_atlaga)
```

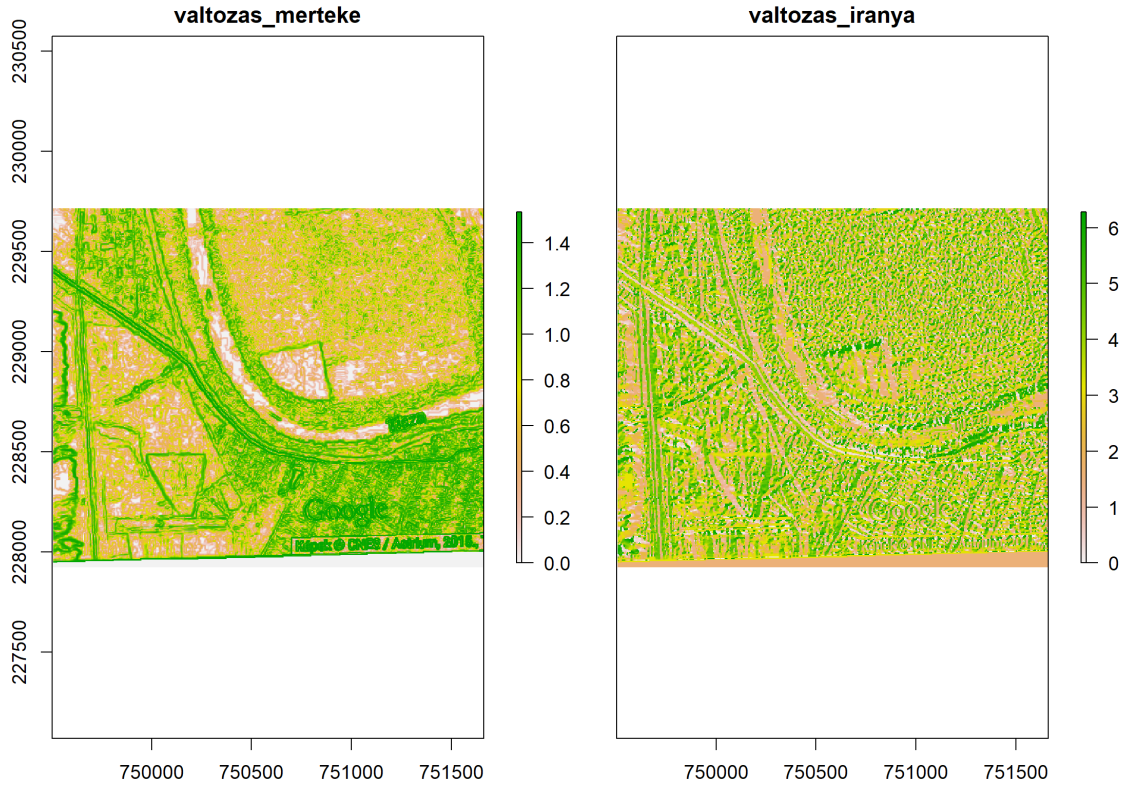
```
[1] 2
```

```
names(valtozasok_atlaga) <- c("valtozas_merteke",
 "valtozas_iranya")
```

Az eredmény 2 réteg, mert a mean() minden rétegpárra egy új réteget adott eredményül.

Haladó raszteralgebra az `overlay()` függvénnyel

```
plot(valtozasok_atlaga)
```

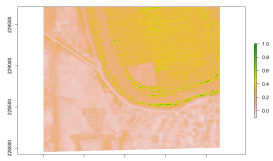


1. feladat (órai)

- Nevezd át a “tisza_kanyar” nevű háromrétegű raszter rétegeit “piros”, “infra” és “uv” nevekre.
- Számold ki egyszerű műveletekkel egy új raszterbe az NDVI értékét. Az NDVI (**n**ormalized **d**ifference **v**egetation **i**ndex) a távérzékelésben használt vegetációborítási mérőszám, eképpen számítható:

(infra - piros) / (infra + piros)

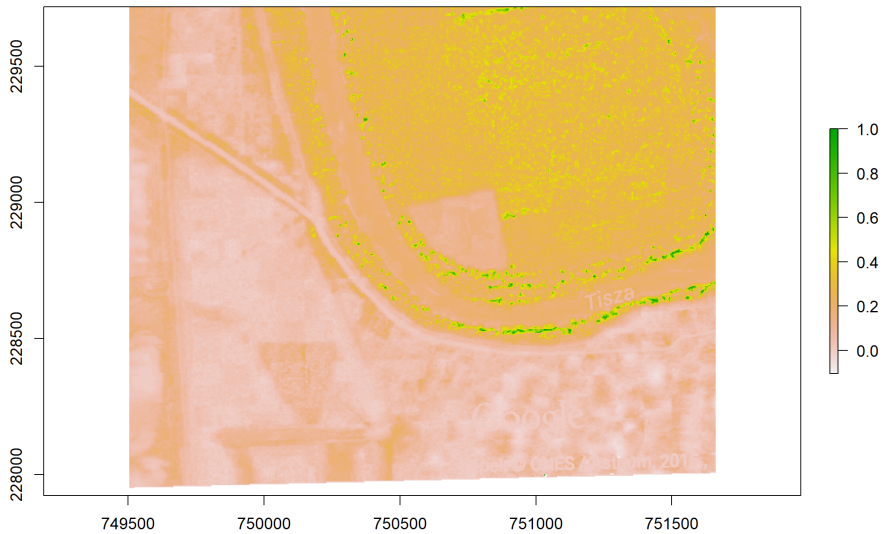
- Jelenítsd meg az NDVI-t.
- Egyszerű függvényhívással (tehát nem a `calc()`-kal/`overlay()`-jel!) számold ki egy egyrétegű raszterbe, hogy a “tisza_kanyar” raszter egyes celláiban mi a három színérték minimuma.
- Jelenítsd meg az eredményt a szürke színskála (`gray.colors()`) tíz színét alkalmazva.



1. feladat (órai) - megoldás

```
names(tisza_kanyar) <- c("piros", "infra", "UV")
ndvi <- (tisza_kanyar$infra - tisza_kanyar$piros) /
 (tisza_kanyar$infra + tisza_kanyar$piros)
```

```
plot(ndvi)
```



1. feladat (órai) - megoldás

```
szinek_minimuma <- min(tisza_kanyar)
plot(szinek_minimuma, col = gray.colors(10))
```

2. feladat (házi)

- Ismerkedj meg a `diff()` függvénnyel, amely egy n elemű vektort vár bemenetként, és n - 1 elemű vektorral tér vissza, amely minden egymást követő szám különbségét tartalmazza.

```
diff(x = c(3, 6, 7, 9, 8))
```

```
[1]  3  1  2 -1
```

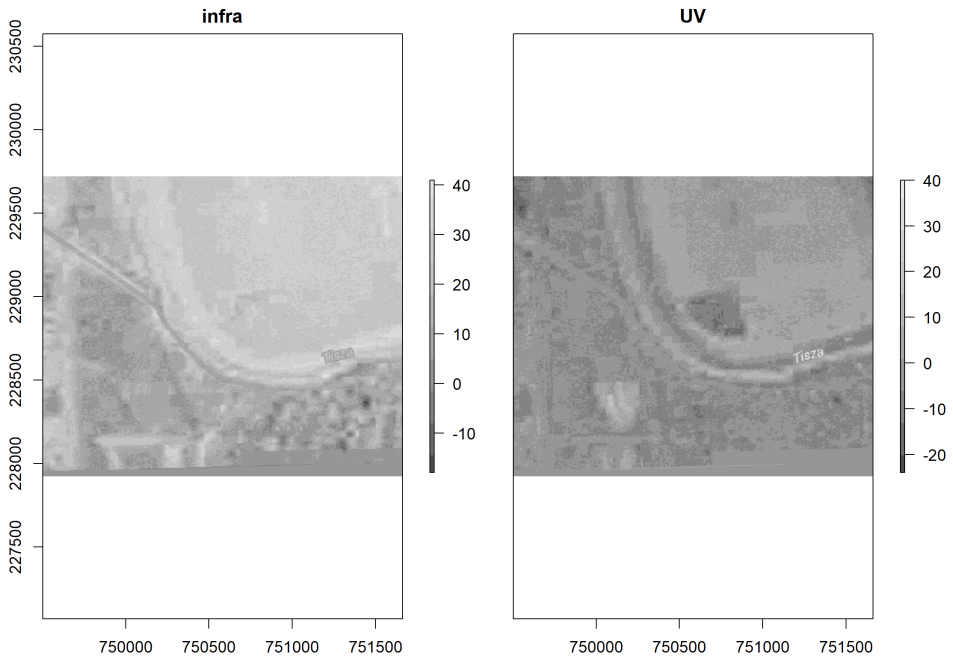
- Számítsd ki a "tisza_kanyar" háromrétegű raszter egymást követő rétegeinek eltérését egy kételemű raszterbe.
- Jelenítsd meg az eredményt a szürke színskála (`gray.colors()`) tíz színét alkalmazva.
- Ezután számítsd ki az "esztergom_felszinboritas", "esztergom_homerseklet" és "esztergom_dem" raszterek átlagát az `overlay()` függvénnyel.
- Vígasztalódás gyanánt, hogy ennek mégoly csekély értelme sem volt, továbbá a `calc()` erre megfelelőbb lett volna, jelenítsd meg az eredményt.

2. feladat (házi) - megoldás

```
diff(x = c(3, 6, 7, 9, 8))
```

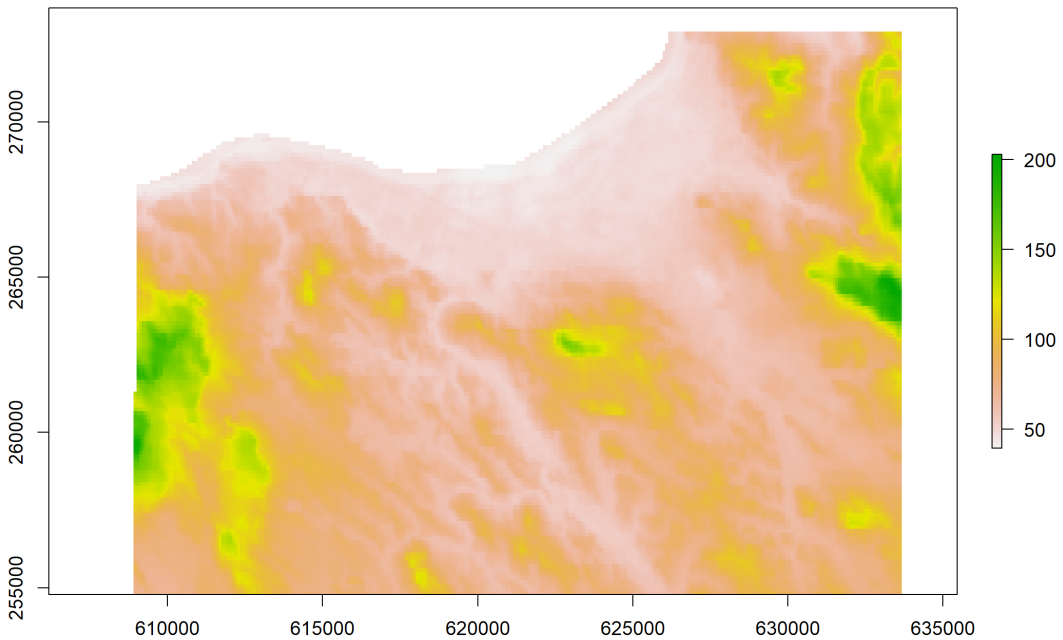
```
[1]  3  1  2 -1
```

```
elteresek <- calc(x = tisza_kanyar, fun = diff)
plot(elteresek, col = gray.colors(10))
```



2. feladat (házi) - megoldás

```
atlag <- overlay(x = esztergom_felszinboritas, y =
 esztergom_homerseklet, esztergom_dem, fun = mean)
plot(atlag)
```



Section 2

Statisztikák

Statisztikák

```
cellStats(x, stat = "mean", na.rm = TRUE)
```

- raszteralgebrával/`calc()`-kal/`overlay()`-jel mindig a cella volt az alapegység, és a rétegek között tudtunk statisztikát számolni
- a `cellStats` fordítva működik: rétegenként vonja össze a cellákat, és a réteghez tartozó összes cellára számol statisztikát
- `x`: bementi raszter, jellemzően többrétegű
- `stat`: a statisztikai függvény
- `na.rm`: elhagyja-e az ismeretlen cellákat? (alapértelmezett: igen)

Statisztikák

Cellaértékek átlaga rétegenként:

```
cellStats(x = tisza_kanyar, stat = mean, na.rm = TRUE)
```

```
   piros    infra       UV
66.72334 83.29701 80.42847
```

Az eredmény egy háromelemű számvektor.

Statisztikák

Minimumok képzése, ismeretlenek elhagyása nélkül:

```
cellStats(x = stack(esztergom_dem, esztergom_homerseklet,
 esztergom_felszinboritas), stat = min, na.rm = FALSE)
```

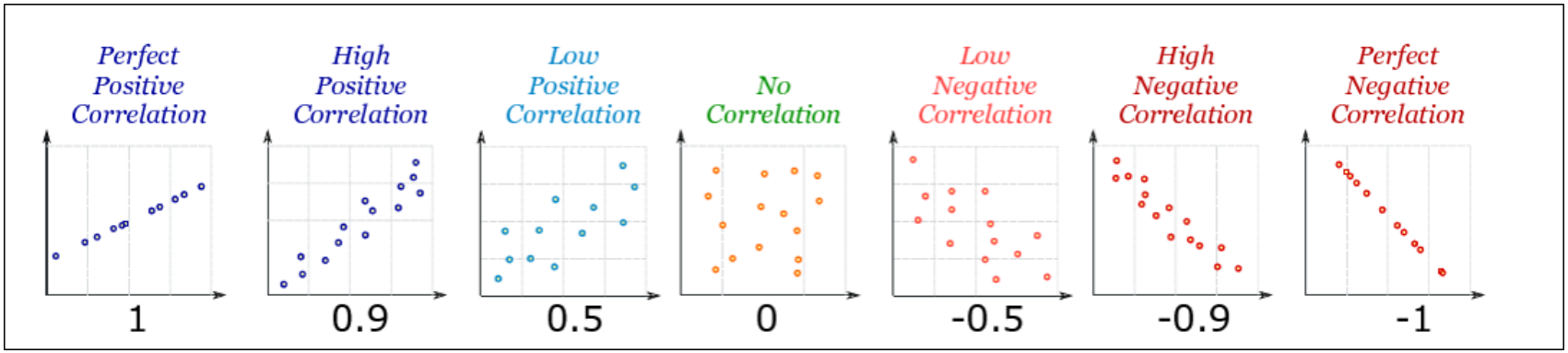
```
[1] NA NA  2
```

Jobb inkább az alapértelmezett `na.rm = TRUE`:

```
cellStats(x = stack(esztergom_dem, esztergom_homerseklet,
 esztergom_felszinboritas), stat = min, na.rm = TRUE)
```

```
[1] 99.000000  2.951776  2.000000
```

Rétegek közötti korreláció



layerStats(x, stat, na.rm = FALSE)

- a rétegek között Pearson-féle korreláció kiszámítására alkalmas
- `x`: többrétegű raszter
- `stat`: a korrelációszámítás módja, esetünkben: `"pearson"`
- `na.rm`: elhagyja-e az ismeretlen cellákat? (alapértelmezett: nem)

`pairs(x)`

- összetett korrelációs ábrát készít

Rétegek közötti korreláció

```
layerStats(x = tisza_kanyar, stat = "pearson", na.rm =
 FALSE)
```

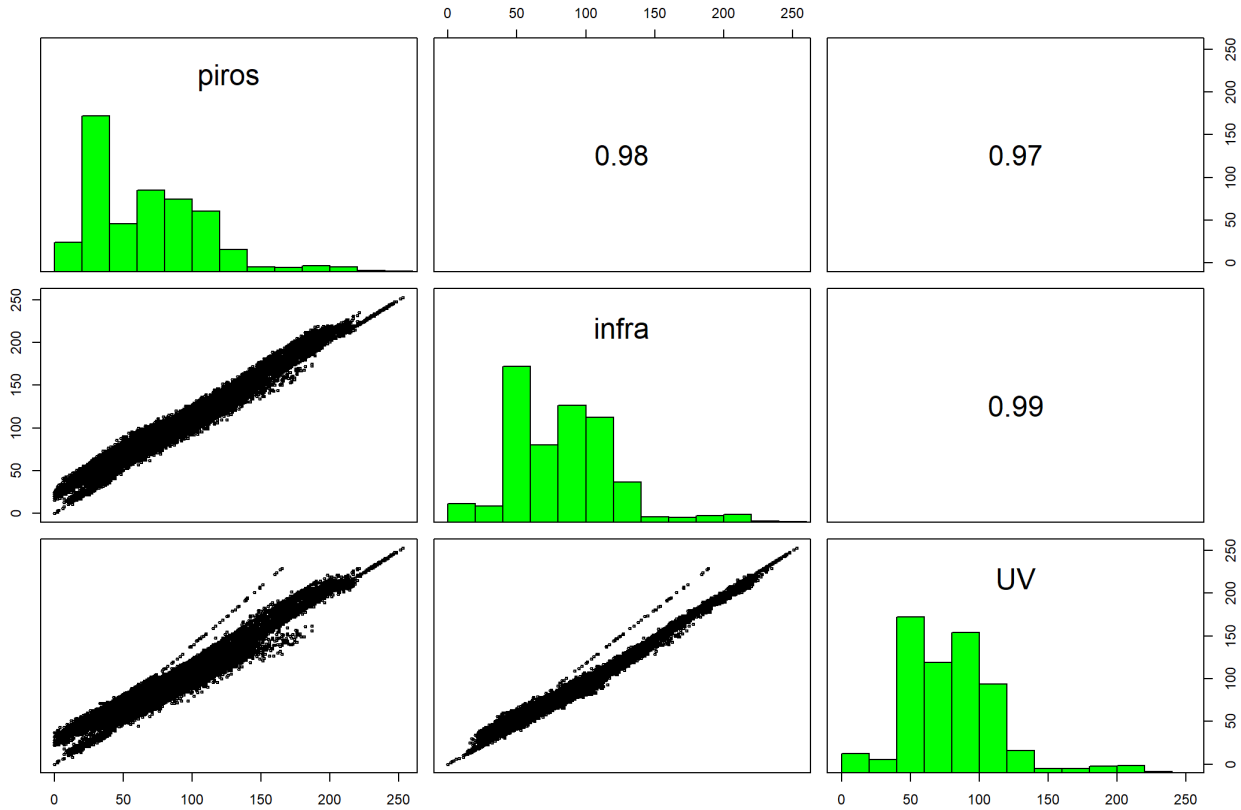
```
$`pearson correlation coefficient`
          piros     infra        UV
piros 1.0000000 0.9757438 0.9655542
infra 0.9757438 1.0000000 0.9928011
UV    0.9655542 0.9928011 1.0000000

$mean
   piros    infra       UV
66.72334 83.29701 80.42847
```

Korrelációs mátrixot és a rétegek átlagát kapjuk vissza.

Rétegek közötti korreláció

```
pairs(tisza_kanyar)
```



pontfelhő + hisztogram + korrelációs együttható

Statisztikai ábrázolások a `rasterVis` csomaggal

`histogram(x)`

- hisztogramokat készít rétegenként
- egy hisztogram az adott réteg összes cellájának eloszlását mutatja

`densityplot(x)`

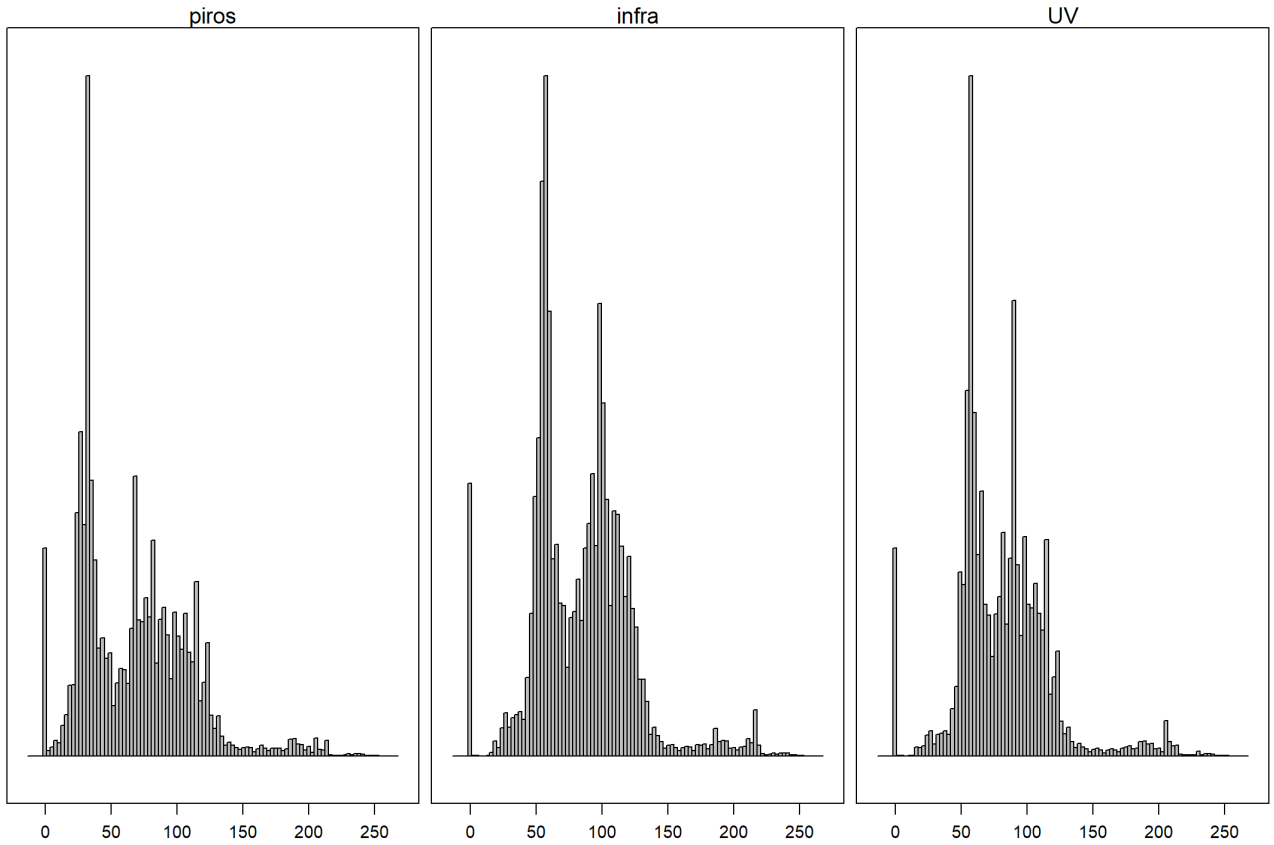
- egymásra lapolt eloszlásgörbéket rajzol rétegenként
- egy görbe az adott réteg összes cellájának eloszlását mutatja

`bwplot(x, violin = TRUE)`

- boxplotot és hegedűgörbét rajzol rétegenként
- `x`: bemeneti (többrétegű) raszter
- `violin`: a boxplotra rajzoljon hegedűgörbét is? (alapértelmezett: igen)

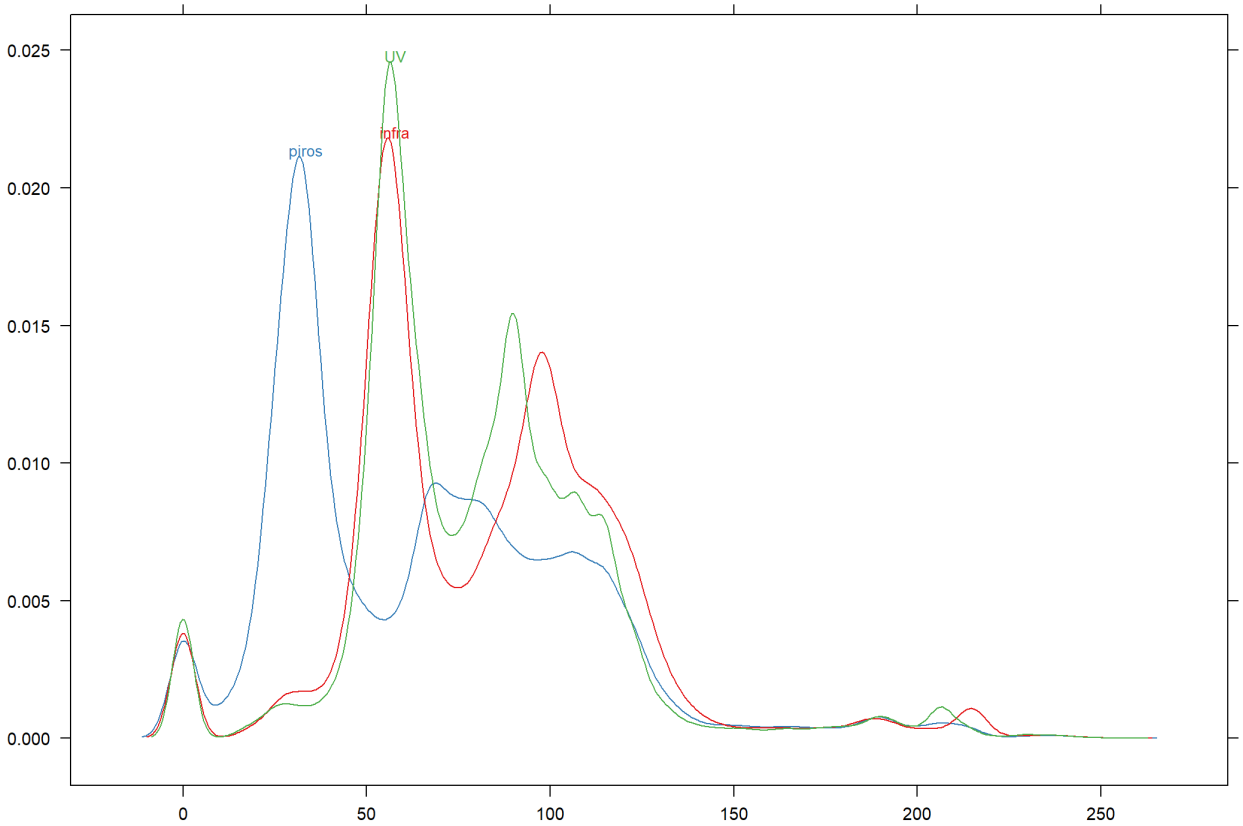
Statisztikai ábrázolások a rasterVis csomaggal

```
histogram(tisza_kanyar)
```



Statisztikai ábrázolások a rasterVis csomaggal

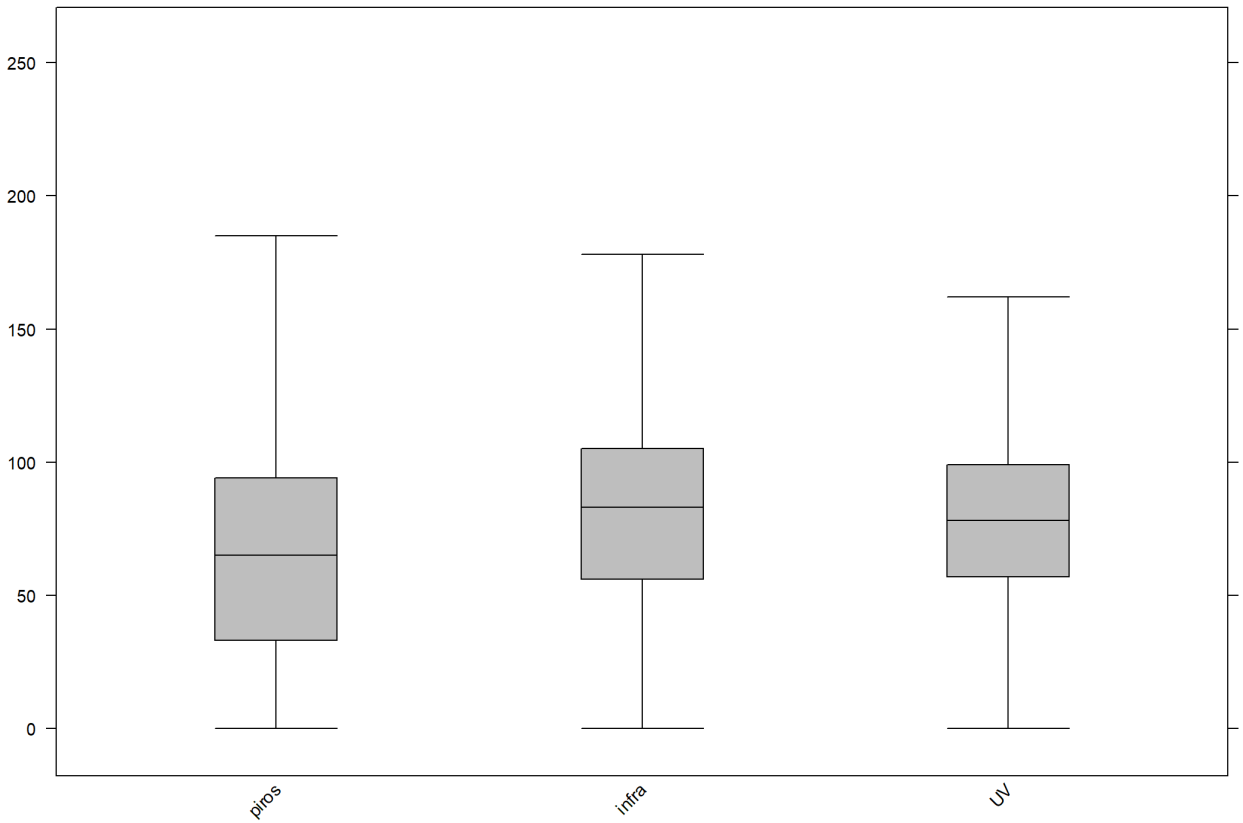
```
densityplot(tisza_kanyar)
```



Statisztikai ábrázolások a rasterVis csomaggal

Csak boxplot:

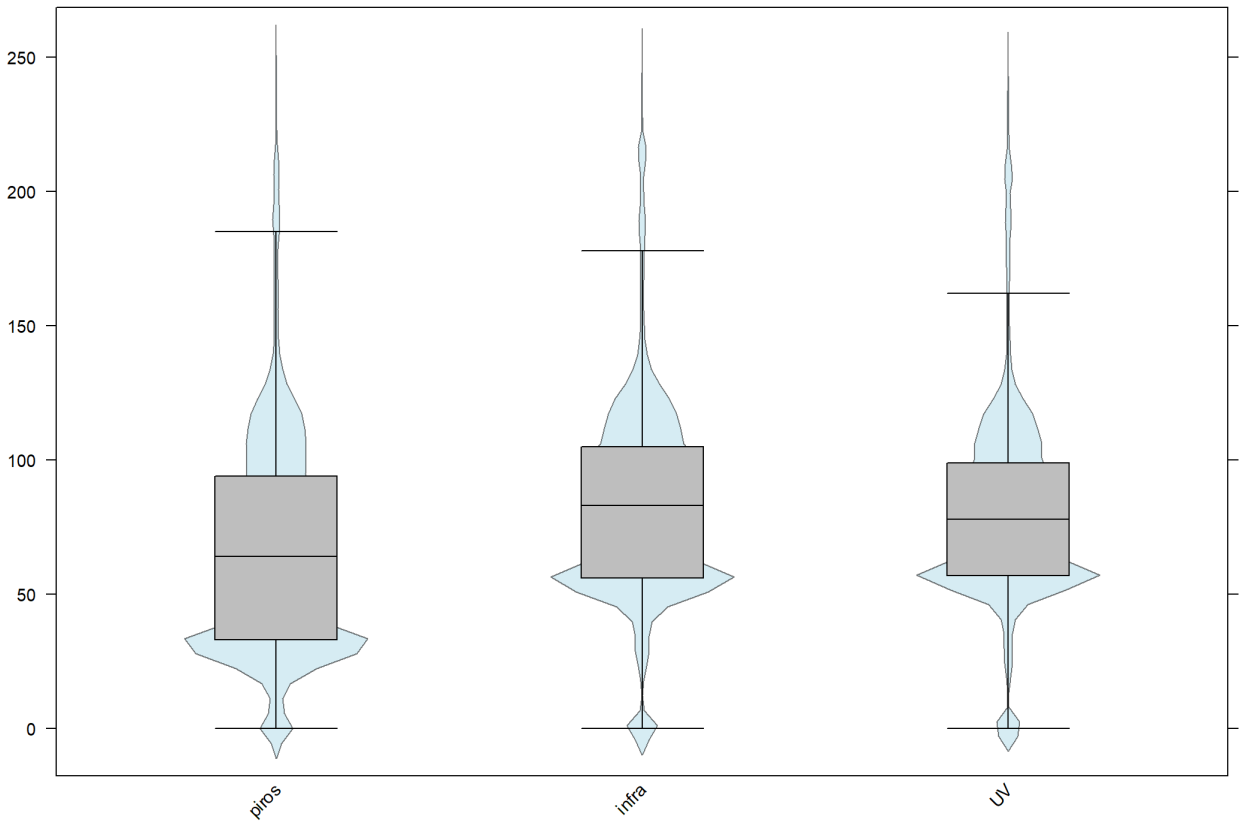
```
bwplot(x = tisza_kanyar, violin = FALSE)
```



Statisztikai ábrázolások a rasterVis csomaggal

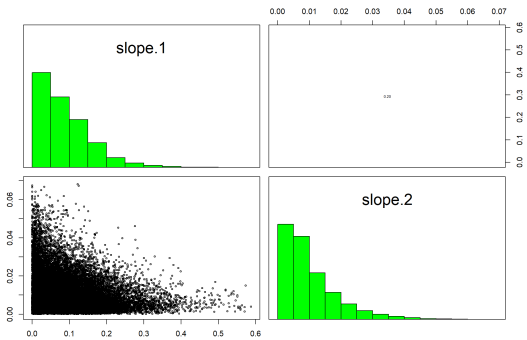
Boxplot és hegedűgörbe:

```
bwplot(x = tisza_kanyar, violin = TRUE)
```



3. feladat (házi)

- Számold ki a “tisza_kanyar” nevű raszter minden rétegére a cellák maximumát úgy, hogy az ismeretlen cellákat is beleveszed az elemzésbe.
- Számold ki a “valtozasok” nevű raszter minden rétegére a cellák maximumát úgy, hogy az ismeretlen cellákat kihagyod a számításból.
- Számold ki a “valtozasok” nevű raszter rétegei közötti Pearson-féle korrelációt úgy, hogy az ismeretlen cellákat kihagyod a számításból.
- Jelenítsd meg egy összetett korrelációs ábrán a “valtozasok” rétegei közötti összefüggést.
- Jelenítsd meg a “valtozasok” egyes rétegeinek boxplotját úgy, hogy hegedűgörbét is raksz a boxplotokra.



3. feladat (házi) - megoldás

```
cellStats(x = tisza_kanyar, stat = max, na.rm = FALSE)
```

```
[1] 254 254 254
```

```
cellStats(x = valtozasok, stat = max, na.rm = TRUE)
```

```
[1] 0.58800031 0.06916938
```

3. feladat (házi) - megoldás

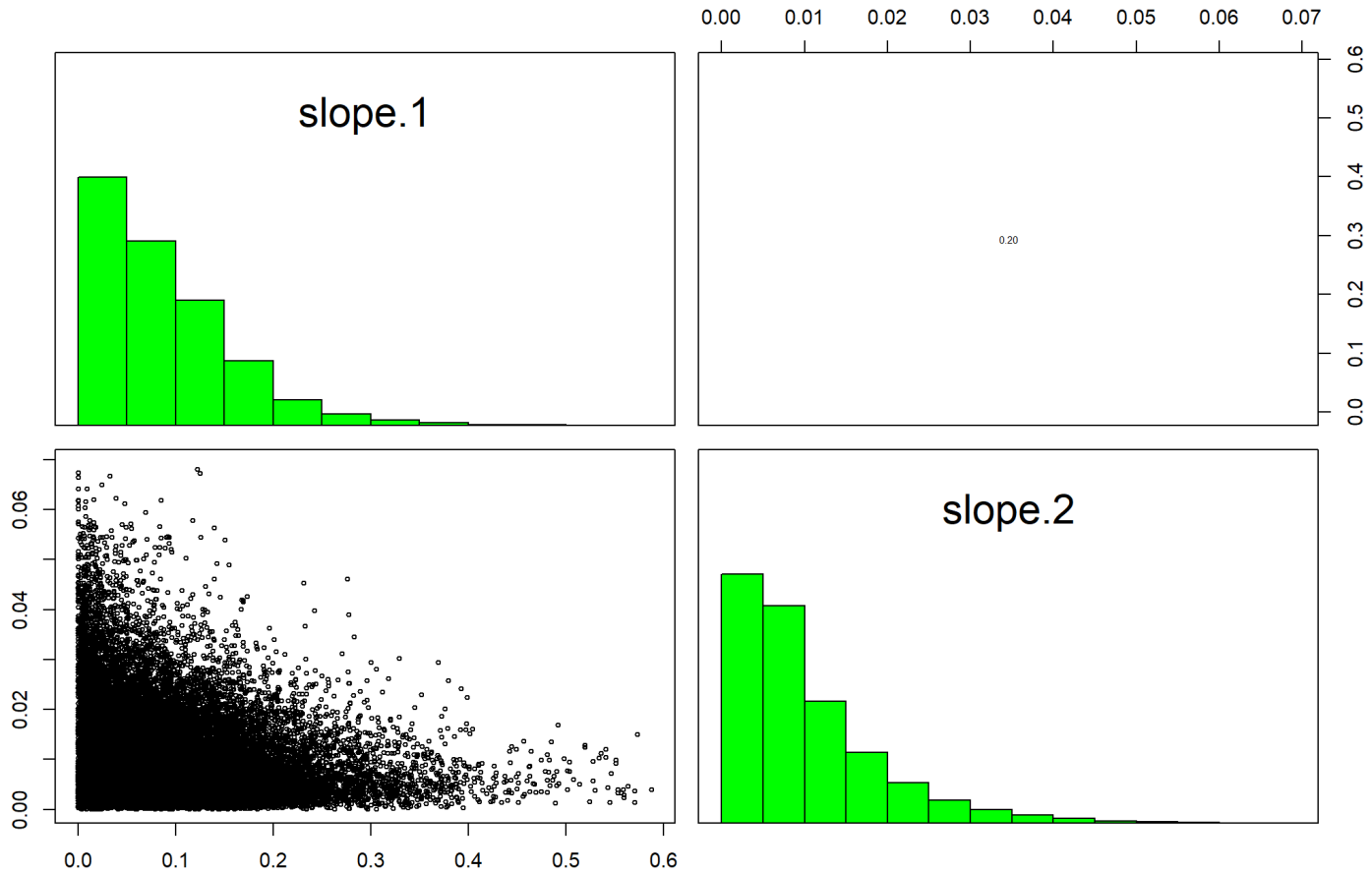
```
layerStats(x = valtozasok, stat = "pearson", na.rm = TRUE)
```

```
$`pearson correlation coefficient`
           slope.1    slope.2
slope.1  1.0000000 -0.1919733
slope.2 -0.1919733  1.0000000

$mean
   slope.1    slope.2
0.08727638 0.01030342
```

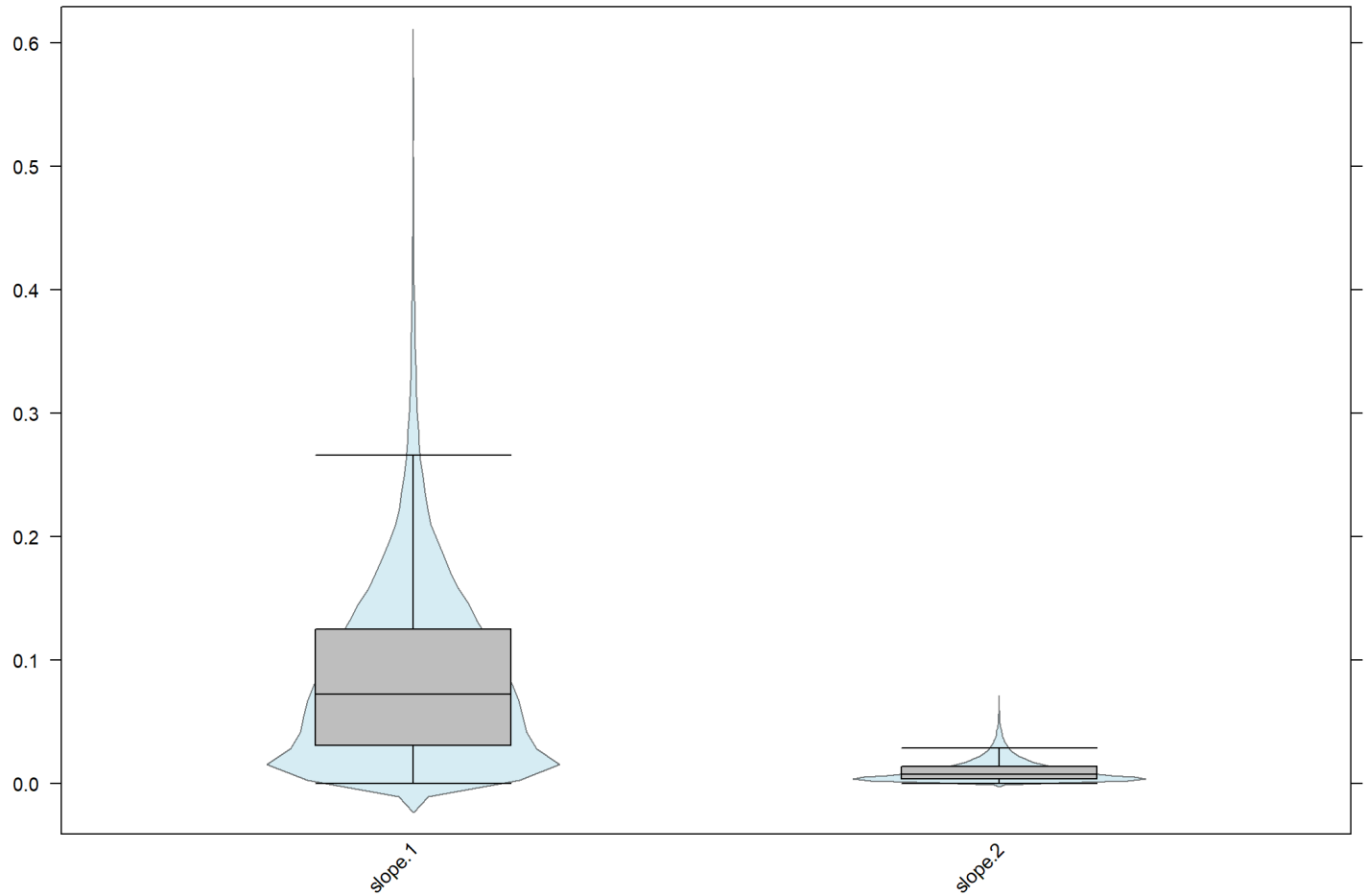
3. feladat (házi) - megoldás

```
pairs(valtozasok)
```



3. feladat (házi) - megoldás

```
bwplot(x = valtozasok, violin = TRUE)
```



Section 3

Térbeli autokorreláció

Térbeli autokorreláció

Autokorreláció:

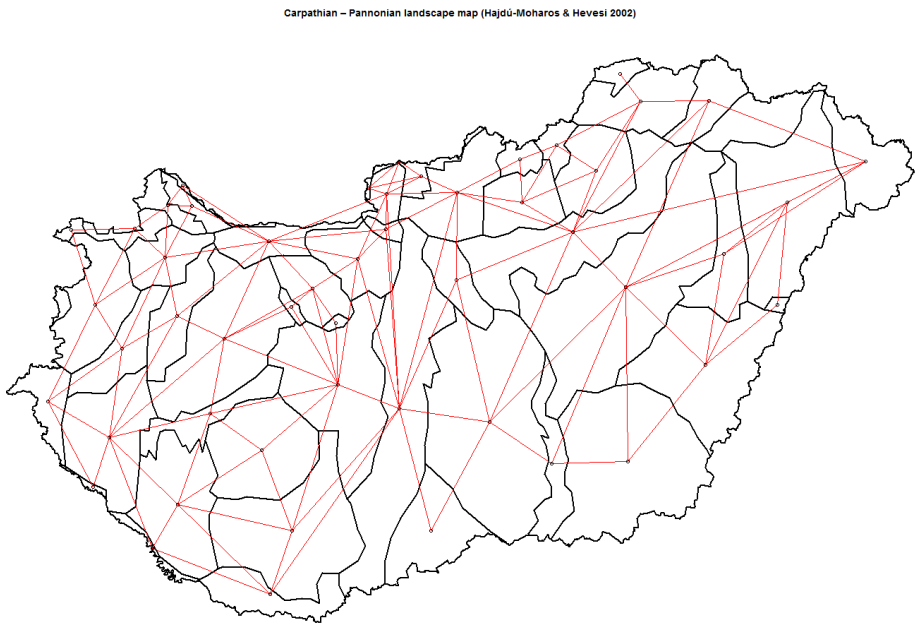
- önmagával (értsd: pl. domborzat a domborzattal) mennyire korrelál

Térbeli autokorreláció:

- egy földrajzi pont (pl. rasztercella) értéke mennyire becsülhető meg a környező pontok/cellák értékei alapján

Térbeli autokorreláció

Raszterekre és vektorokra egyaránt számítható (most csak raszterre mutatom be).
Vektorok esetén előzetesen szomszédsági gráfot kell készíteni → `spdep` csomag

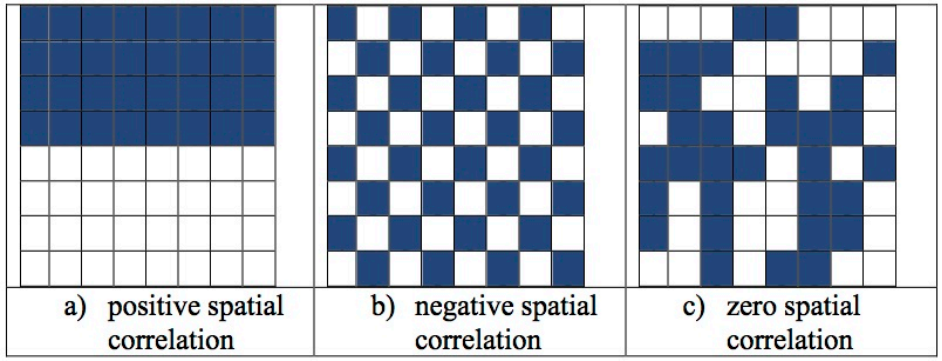


Térbeli autokorreláció

Térbeli autokorreláció típusai

- több metrika létezik: Moran-féle I, Geary-féle C
- mindegyiket lehet globálisan (az egész raszterre) és lokálisan (kb. mint a mozó ablak) számolni
- globális esetben az eredmény egy szám, lokális esetben egy raszter

Térbeli autokorreláció



Moran-féle I

- -1 és 1 közötti szám (globális)
- lokális esetben kiléphet az intervallumból
- jellemző esetek:
  - véletlen eloszlású cellák: nem autokorrelált (0)
  - nagy blokkok: pozitívan autokorrelált (+1)
  - szabályosan ismétlődő cellák (sakktáblaszerű): negatívan autokorrelált (-1)
- a Geary-féle C csak pozitív lehet, az iránya pedig fordított

Térbeli autokorreláció

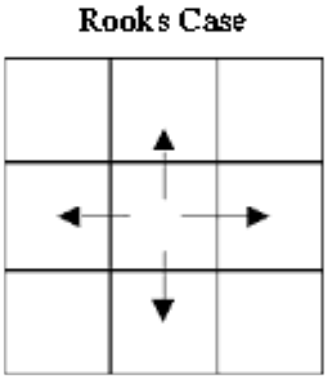
`Moran(x, w)`

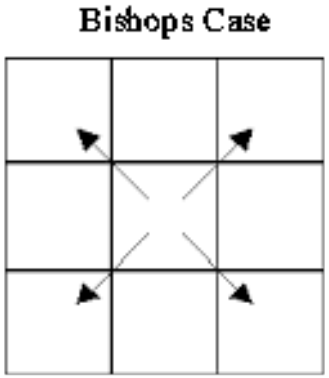
- globális térbeli autokorrelációt számol
- `x`: egyrétegű raszter
- `w`: súlymátrix (alapértelmezett: 3×3-as mátrix csupa 1-essel, de a közepén 0-val, "királynő")
- de a Moran-féle I kiszámításához valójában a "bástya" súlymátrix kellene...
- eredmény: egy szám

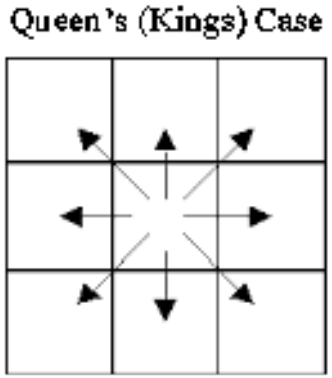
`MoranLocal(x, w)`

- lokális térbeli autokorrelációt számol minden cellára
- `x`, `w`: mint a `Moran()`-nál
- eredmény: egyrétegű raszter

Térbeli autokorreláció







Súlymátrixok

- általában 3×3-asak, de definiálhatunk nagyobbat is
- a sakkbábuk mozgása alapján nevezik el a fő típusokat:
  - bástya: oldalirányba/föl/le
  - futó: átlósan
  - királynő: minden irányban

Térbeli autokorreláció

Hozzunk létre néhány példarasztert!

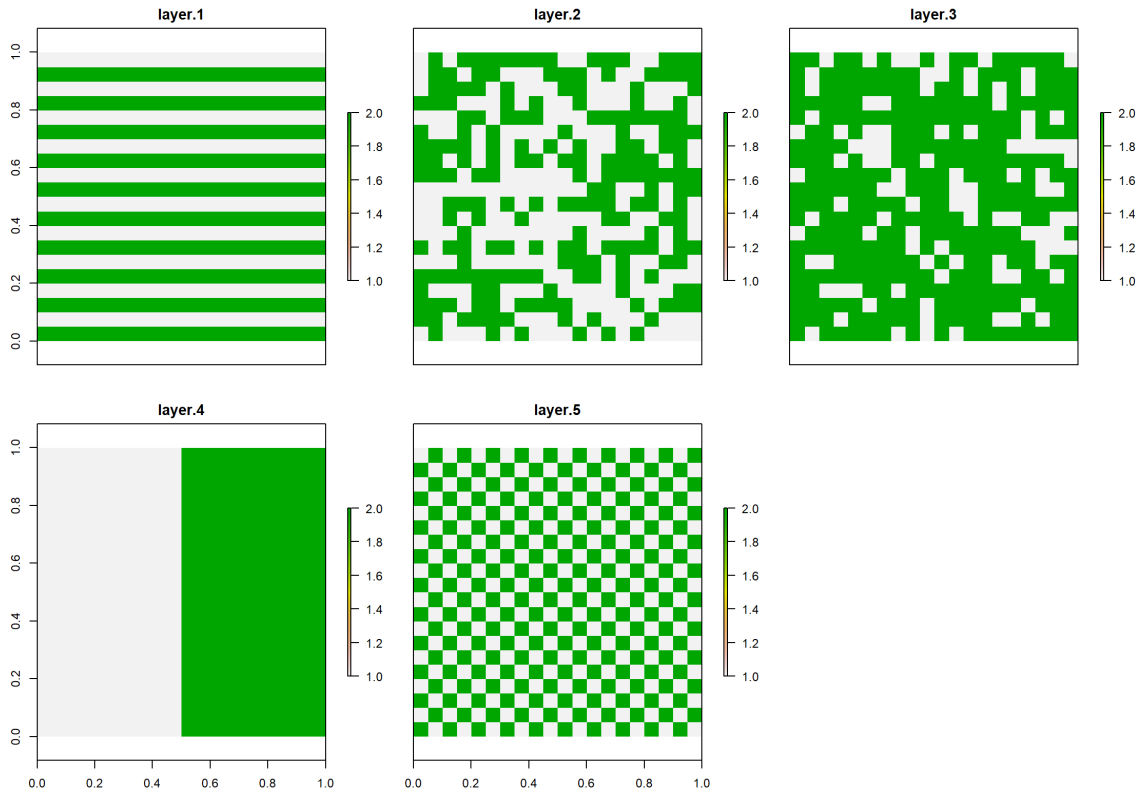
```
set.seed(98765)
szammatrix1 <- matrix(data = 1:2, ncol = 20, nrow = 20)
csikos <- raster(x = szammatrix1)
szamok2 <- sample(x = 1:2, size = 20 * 20, replace = TRUE)
szammatrix2 <- matrix(data = szamok2, ncol = 20, nrow = 20)
veletlen_fele <- raster(x = szammatrix2)
```

Térbeli autokorreláció

```
szamok3 <- sample(x = 1:2, size = 20 * 20, replace = TRUE,
 prob = c(0.25, 0.75))
szammatrix3 <- matrix(data = szamok3, ncol = 20, nrow = 20)
veletlen_negyede <- raster(x = szammatrix3)
szamok4 <- c(rep(x = 1, times = 10 * 20), rep(x = 2, times
 = 10 * 20))
szammatrix4 <- matrix(data = szamok4, ncol = 20, nrow = 20)
ket_tomb <- raster(x = szammatrix4)
szamok5 <- c(rep(x = 1:2, times = 10), rep(x = 2:1, times
 = 10 ))
szammatrix5 <- matrix(data = szamok5, ncol = 20, nrow = 20)
sakktabla <- raster(x = szammatrix5)
```

Térbeli autokorreláció

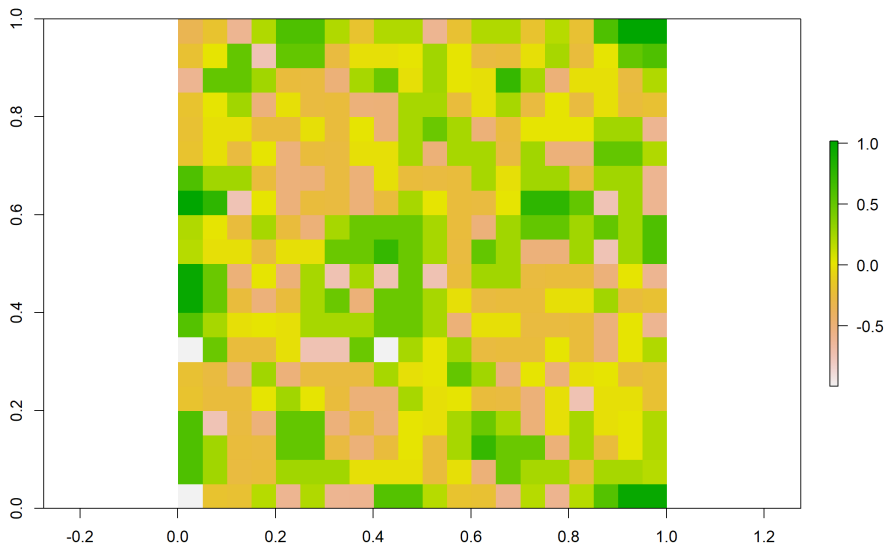
```
plot(stack(csikos, veletlen_fele, veletlen_negyede,
 ket_tomb, sakktabla))
```



Ezeknek eltérő a térbeli autokorrreláltsága.

Térbeli autokorreláció

```
plot(MoranLocal(x = veletlen_fele))
```




```
Moran(x = veletlen_fele)
```

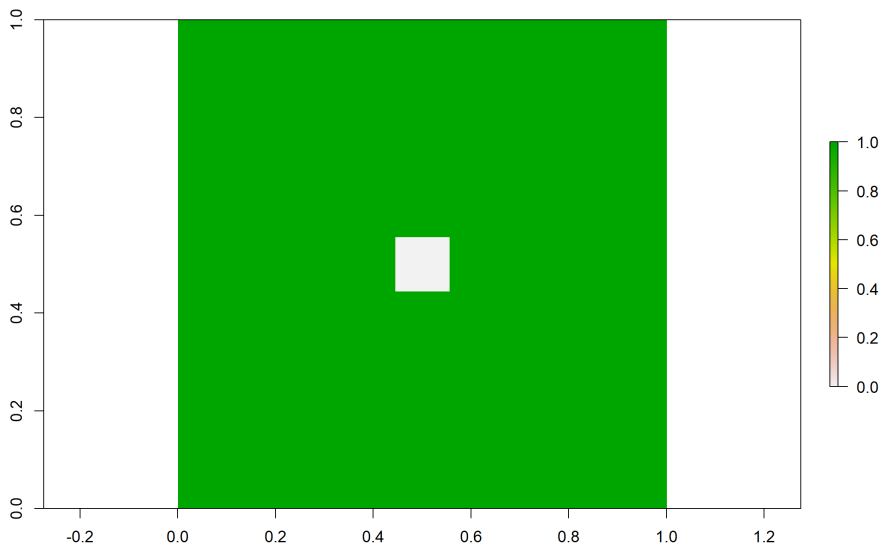
```
[1] -0.01223591
```

Nullához közeli érték: nincs térbeli autokorreláltság.

Térbeli autokorreláció

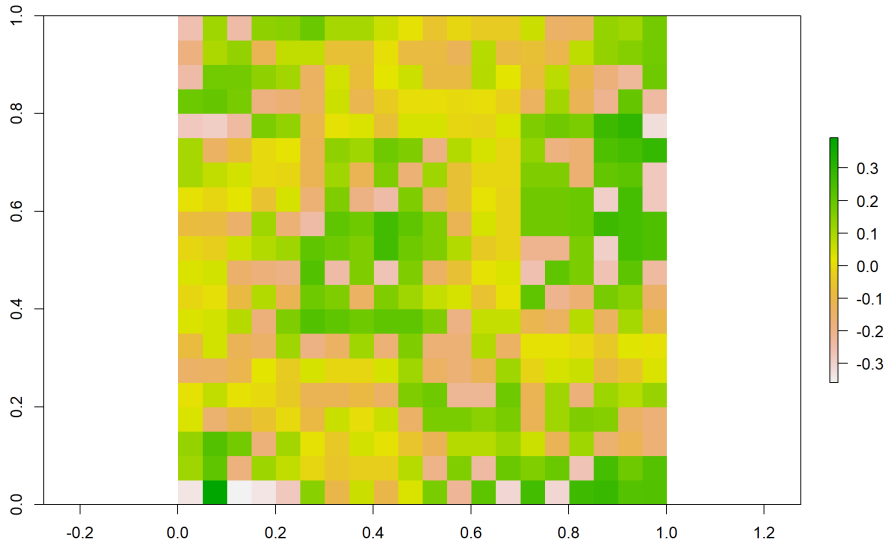
Nézzük meg nagyobb, 9×9-es súlymátrixszal!

```
kiralyno9 <- matrix(data = 1, nrow = 9, ncol = 9)
kiralyno9[5, 5] <- 0
plot(raster(kiralyno9))
```



Térbeli autokorreláció

```
plot(MoranLocal(x = veletlen_fele, w = kiralyno9))
```



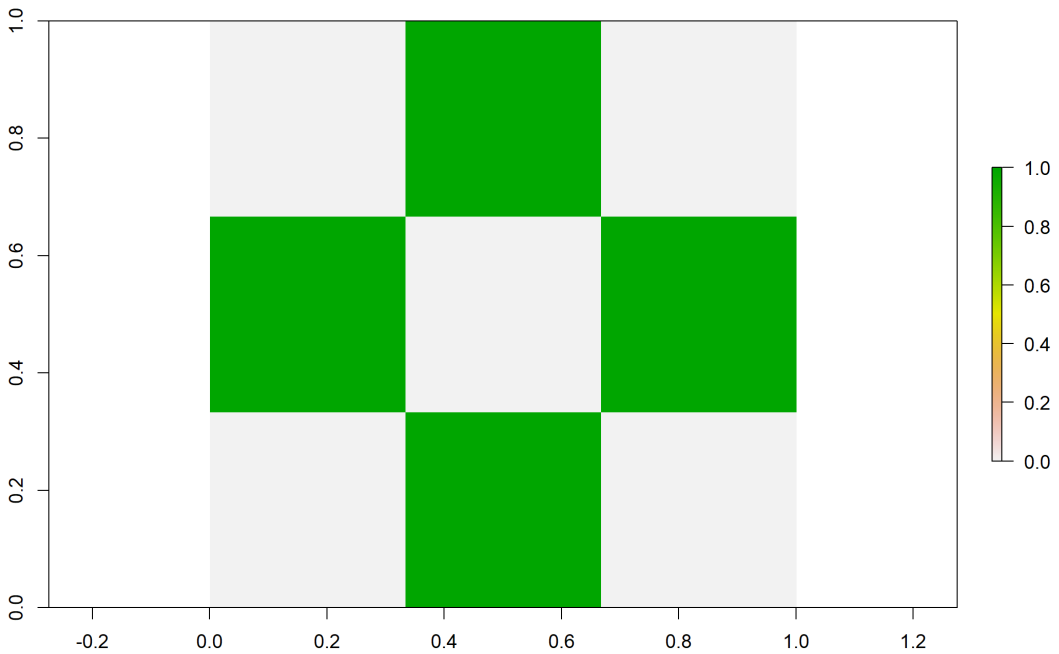

```
Moran(x = veletlen_fele, w = kiralyno9)
```

```
[1] 0.01261555
```

Térbeli autokorreláció

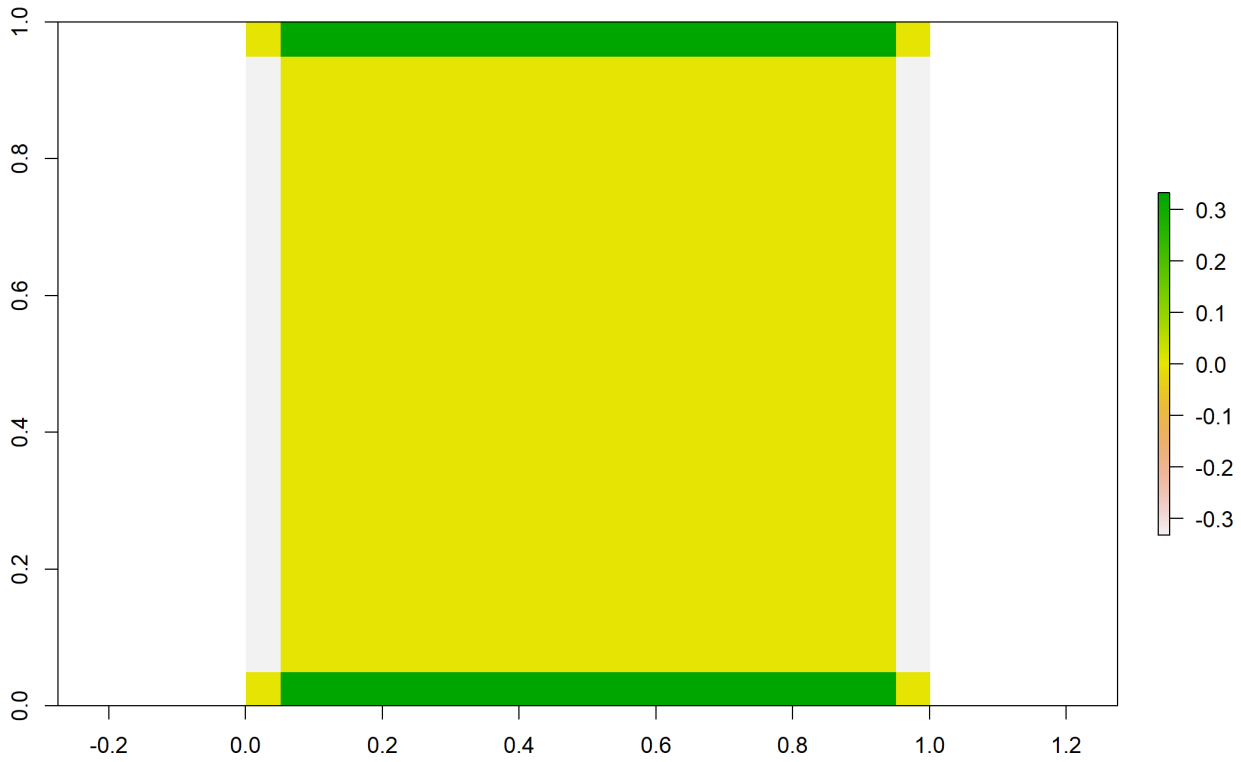
Hozzunk létre 3×3-as “bástya” súlymátrixot, hogy a tényleges Moran-féle térbeli autokorrelációt kiszámolhassuk!

```
bastya3 <- matrix(data = c(0, 1, 0, 1, 0, 1, 0, 1, 0),
 nrow = 3, ncol = 3)
plot(raster(bastya3))
```



Térbeli autokorreláció

```
plot(MoranLocal(x = csikos, w = bastya3))
```

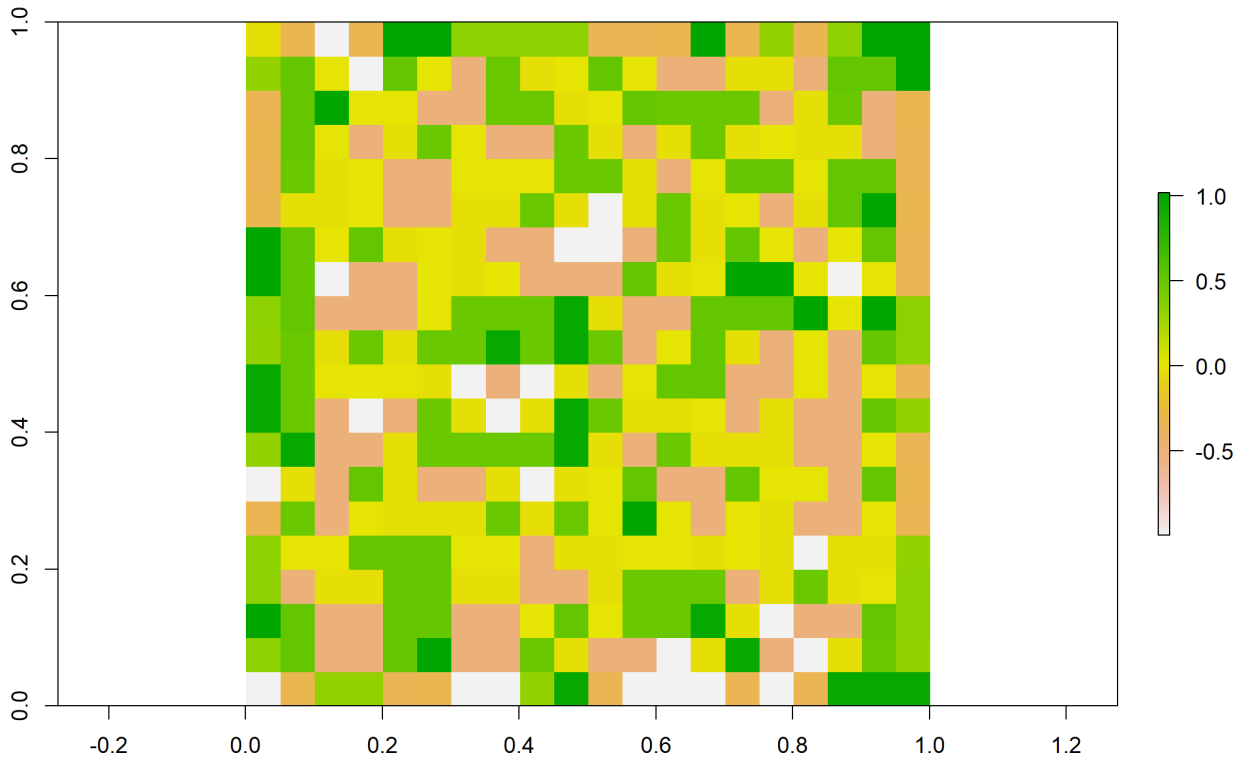



```
Moran(x = csikos, w = bastya3)
```

```
[1] 0
```

Térbeli autokorreláció

```
plot(MoranLocal(x = veletlen_fele, w = bastya3))
```

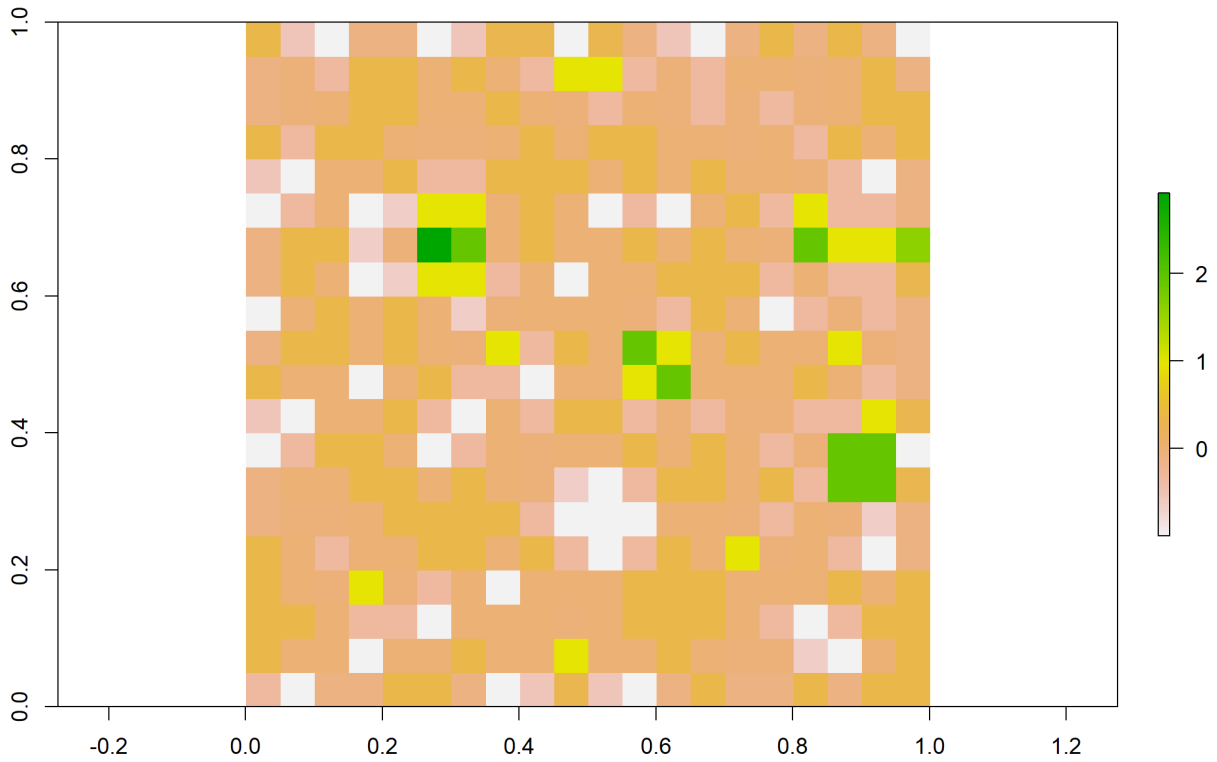



```
Moran(x = veletlen_fele, w = bastya3)
```

```
[1] 0.02360236
```

Térbeli autokorreláció

```
plot(MoranLocal(x = veletlen_negyede, w = bastya3))
```

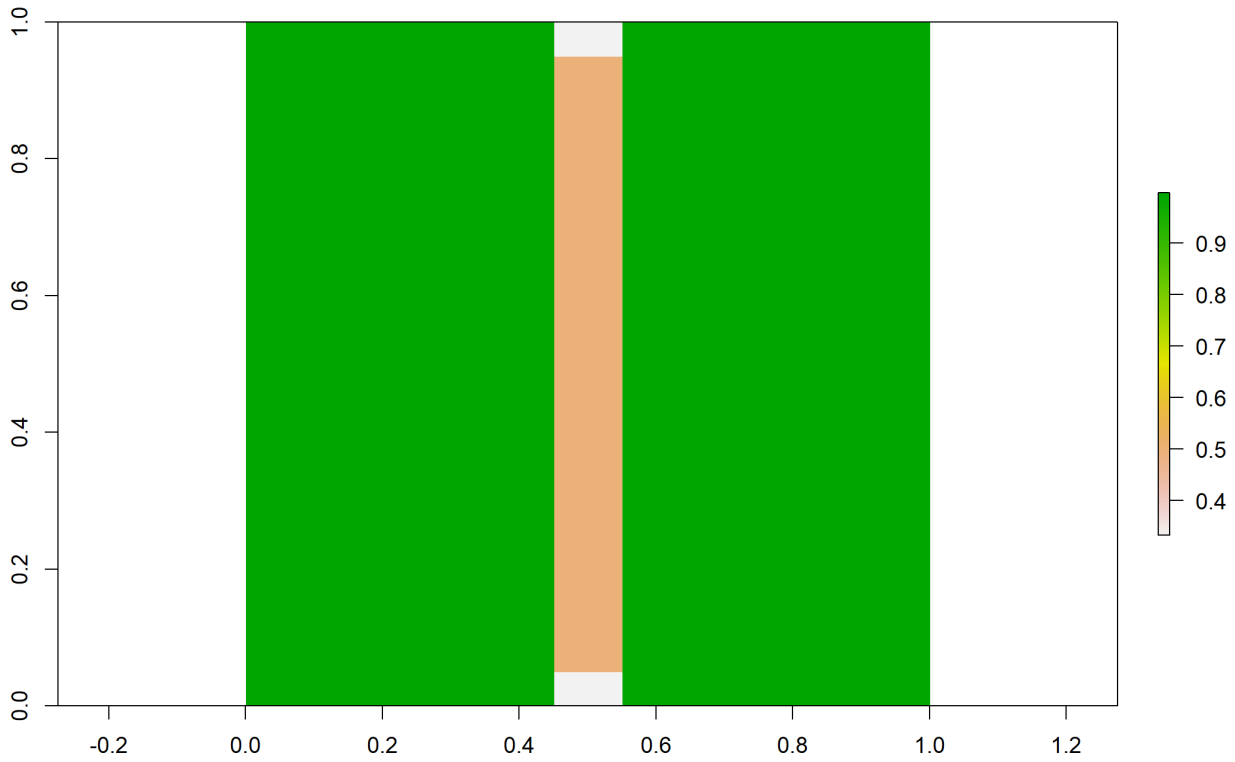



```
Moran(x = veletlen_negyede, w = bastya3)
```

```
[1] 0.02232981
```

Térbeli autokorreláció

```
plot(MoranLocal(x = ket_tomb, w = bastya3))
```

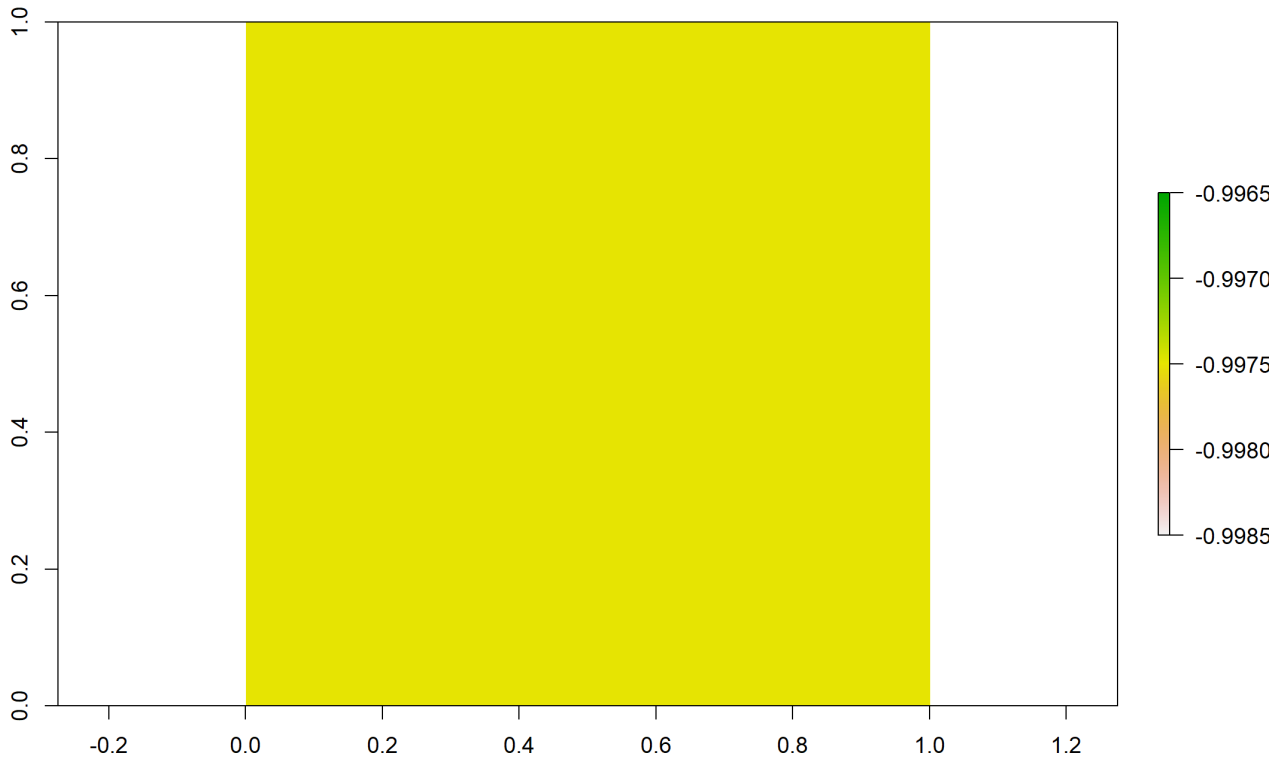



```
Moran(x = ket_tomb, w = bastya3)
```

```
[1] 0.9473684
```

Térbeli autokorreláció

```
plot(MoranLocal(x = sakktabla, w = bastya3))
```




```
Moran(x = sakktabla, w = bastya3)
```

```
[1] -1
```

Térbeli autokorreláció

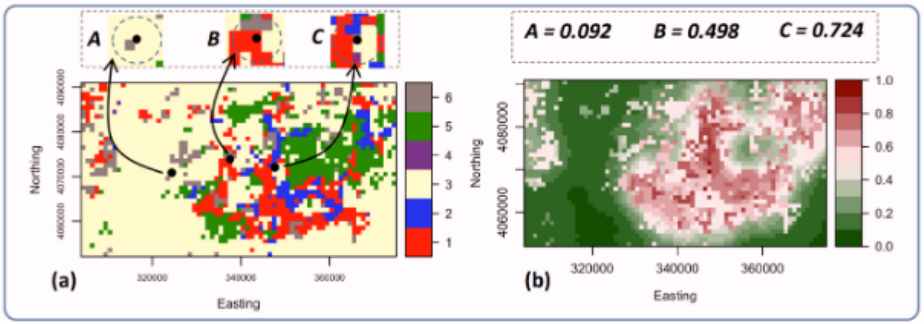
Folytonos felület térbeli autokorreláltsága jellemzően erősen pozitív.
(a 100 m magas réten nem szoktak 500 m magas sziklakiszögellések nőni...)

```
Moran(x = esztergom_dem, w = bastya3)
```

```
[1] 0.9836808
```

Vagyis: tetszőleges ponton állva és a környező magassági értékeket ismerve jól megbecsülhető, hogy milyen magasan állok.

Térbeli autokorreláció kategorikus raszter esetén



Térbeli autokorreláció használhatóságának korlátai

- a térbeli autokorreláció számrasztereken értelmezhető
- technikailag minden raszterre kiszámítható, de kategorikus raszter esetén nincs értelme

Entrópiaalapú helyi térbeli önfüggőség

- ELSA (Entropy-based Local indicator of Spatial Association)
- lokális (a nevében is benne van), tehát rasztert eredményez
- 0: erős térbeli autokorreláció, 1: nincs autokorreláció
- új R-csomag (2019-2020): `elsa`

Térbeli autokorreláció kategorikus raszter esetén

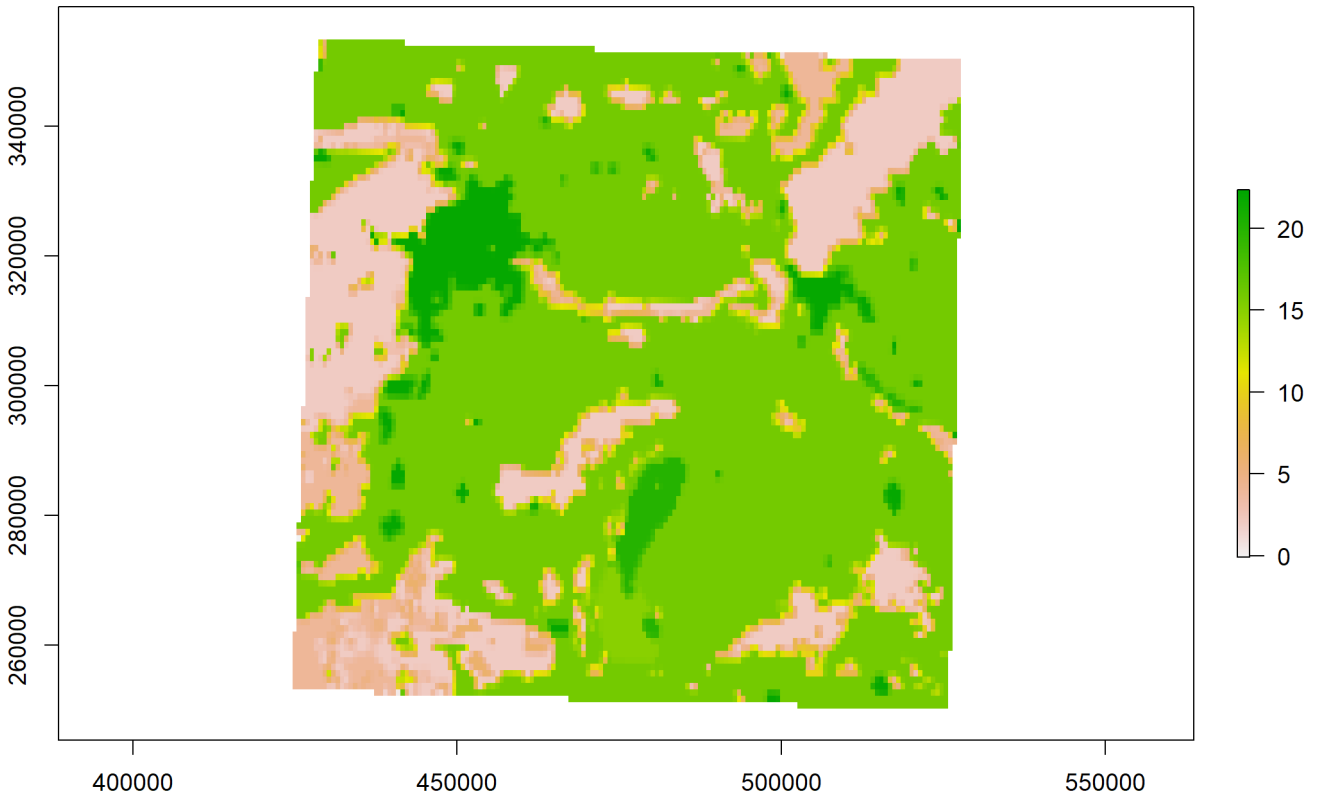
`elsa(x, d, categorical)`

- `x`: egyrétegű raszter
- `d`: távolság (WGS-84 esetén méterben), amin belül számolandó az ELSA
- `categorical`: kategorikus-e az adat? (alapértelmezett: megpróbálja kitalálni)
- ha folytonos az adat, azt kategorizálja. De a módszer elsősorban kategorikus raszterekre lett kidolgozva
- további paraméter a `dif`, amiben a kategóriák közti elvi különbséget lehet definiálni (pl. erdő vs gyep, fenyőerdő vs. lomberdő)
- ezt most nem mutatom be, de többszintű kategóriáknál nagyon hasznos

Térbeli autokorreláció kategorikus raszter esetén

```
becs_felszinboritas <- projectRaster(from =
 felszinboritas[100:200, 50:200, drop = FALSE], crs =
 projection(domborzatmodell))
```

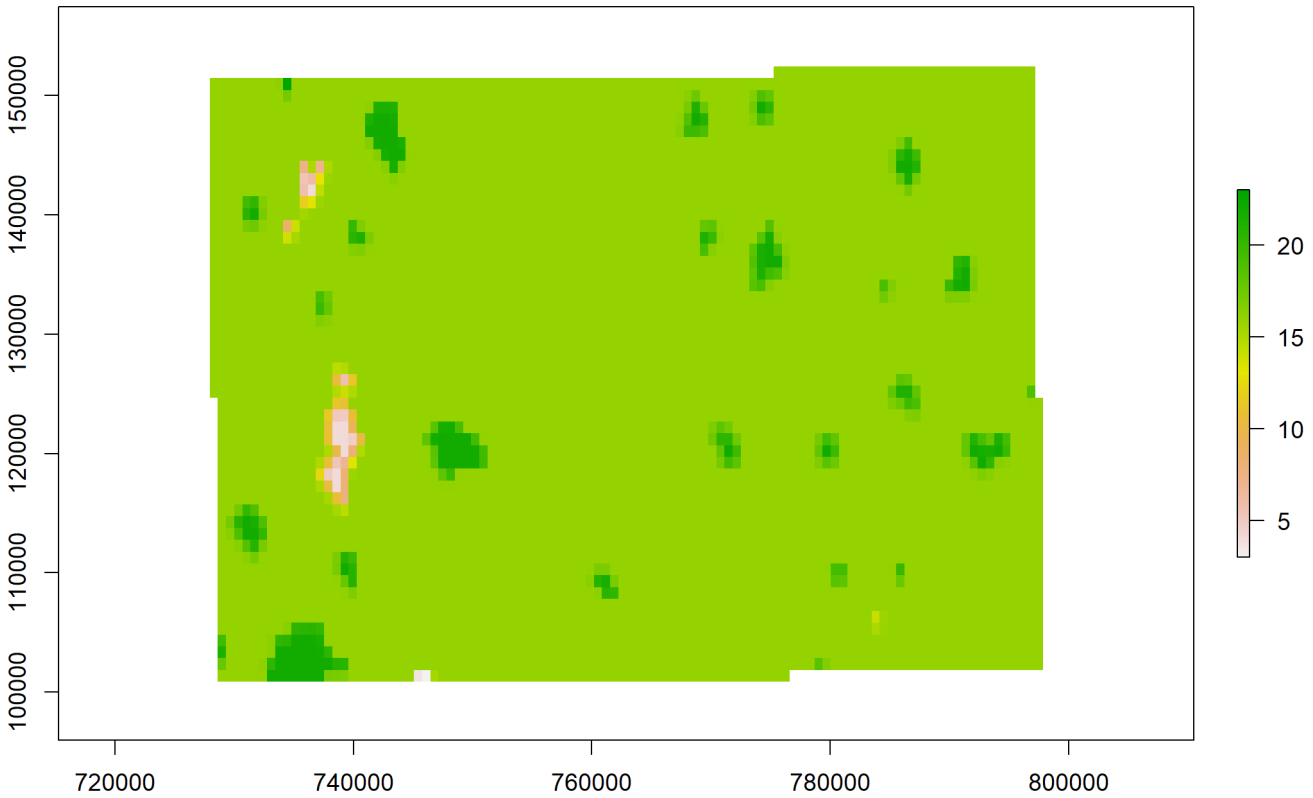
```
plot(becs_felszinboritas)
```



Térbeli autokorreláció kategorikus raszter esetén

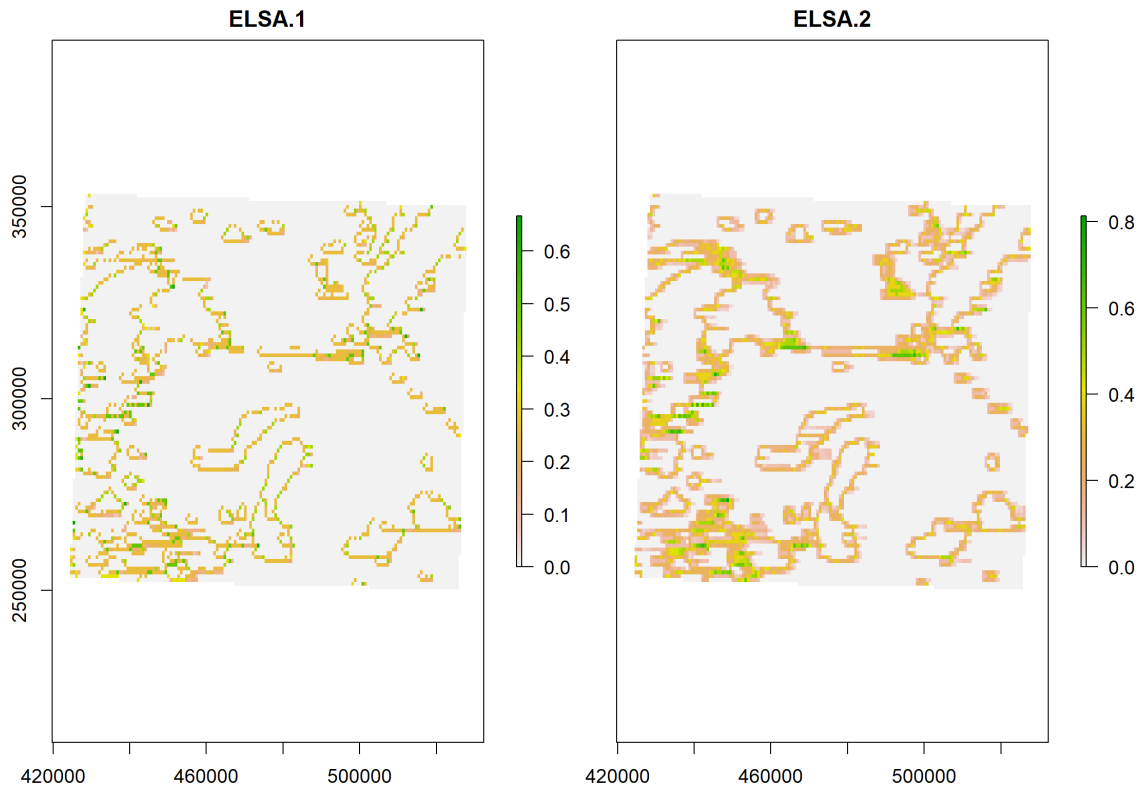
```
alfold_felszinboritas <- projectRaster(felszinboritas
 [300:350, 500:600, drop = FALSE], crs =
 projection(domborzatmodell))
```

```
plot(alfold_felszinboritas)
```



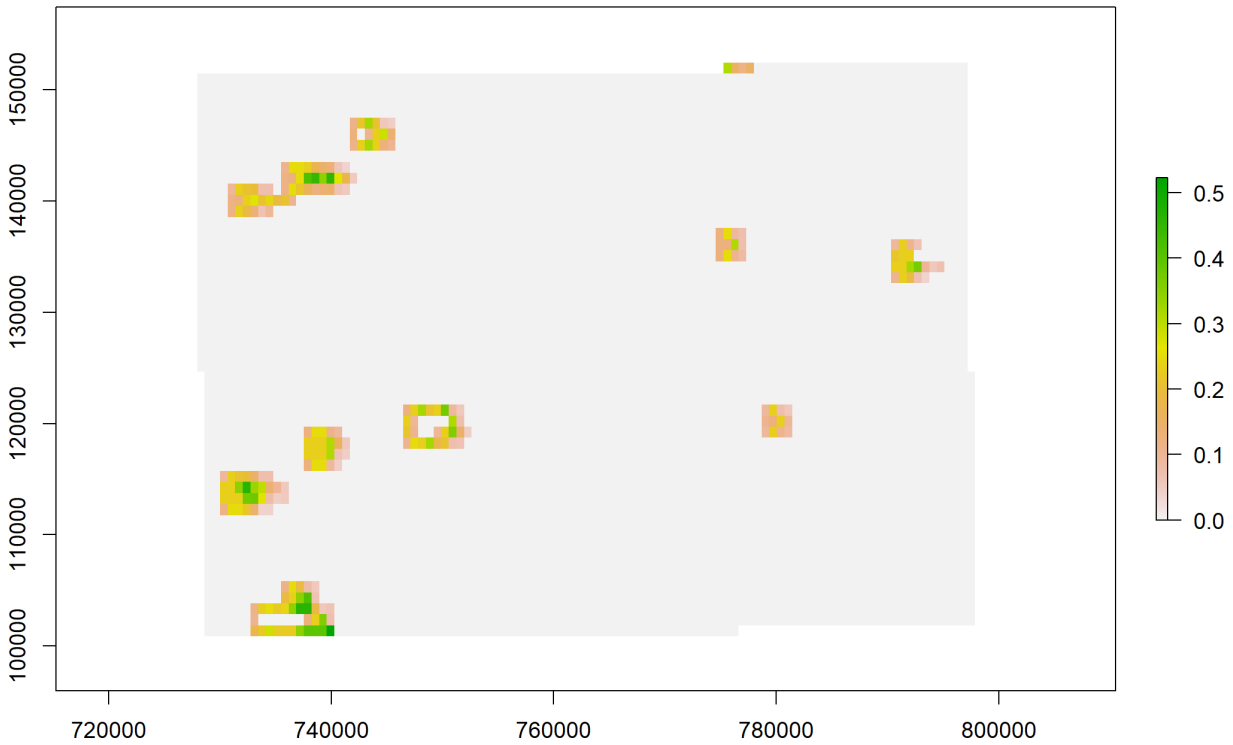
Térbeli autokorreláció kategorikus raszter esetén

```
becs_elsa_700 <- elsa(x = becs_felszinboritas, d = 700,
 categorical = TRUE)
becs_elsa_1000 <- elsa(x = becs_felszinboritas, d = 1000,
 categorical = TRUE)
plot(stack(becs_elsa_700, becs_elsa_1000))
```



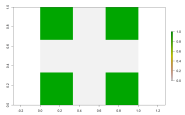
Térbeli autokorreláció kategorikus raszter esetén

```
alfold_elsa_1000 <- elsa(x = alfold_felszinboritas, d =
  1000, categorical = TRUE)
plot(alfold_elsa_1000)
```



4. feladat (órai)

- Hozz létre egy 3×3-as súlymátrixot, ami a futó átlós irányú mozgását imitálja, vagyis a mátrix négy sarkában található 1-es érték.



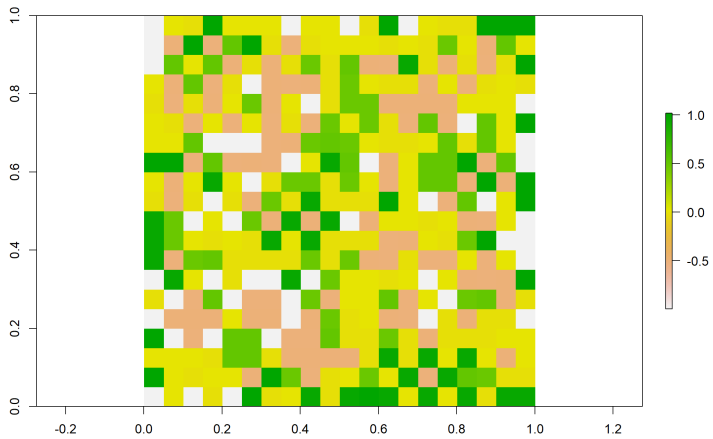
- Számold ki a “veletlen_fele” nevű raszter Moran-féle globális térbeli autokorreláltságát evvel a súlymátrixszal.
- Számold ki és jelenítsd meg ugyanezen raszter lokális térbeli autokorreláltságát evvel a súlymátrixszal.
- Számold ki az “esztergom_homerseklet” nevű raszter Moran-féle globális térbeli autokorreláltságát evvel a súlymátrixszal. Ez alapján mit gondolsz: elmondhatjuk, hogy a cellaértékek jól becsülhetőek a környező cellaértékek ismeretében?
- Számold ki, majd jelenítsd meg az “alfold_felszinboritas” nevű kategorikus raszter 800 méter sugarú körben értelmezett lokális, entrópiaalapú térbeli önfüggésének mértékét.

4. feladat (órai) - megoldás

```
futo3 <- matrix(data = c(1, 0, 1, 0, 0, 0, 1, 0, 1), nrow
 = 3, ncol = 3)
Moran(x = veletlen_fele, w = futo3)
```

```
[1] -0.0499604
```

```
plot(MoranLocal(x = veletlen_fele, w = futo3))
```



4. feladat (órai) - megoldás

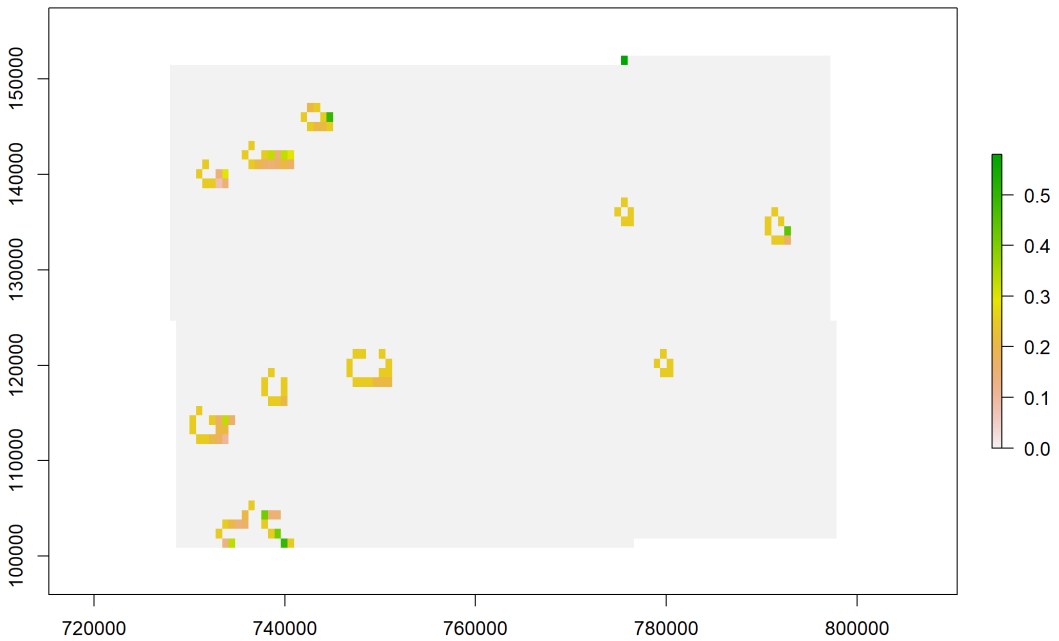
```
Moran(x = esztergom_homerseklet, w = futo3)
```

```
[1] 0.9310802
```

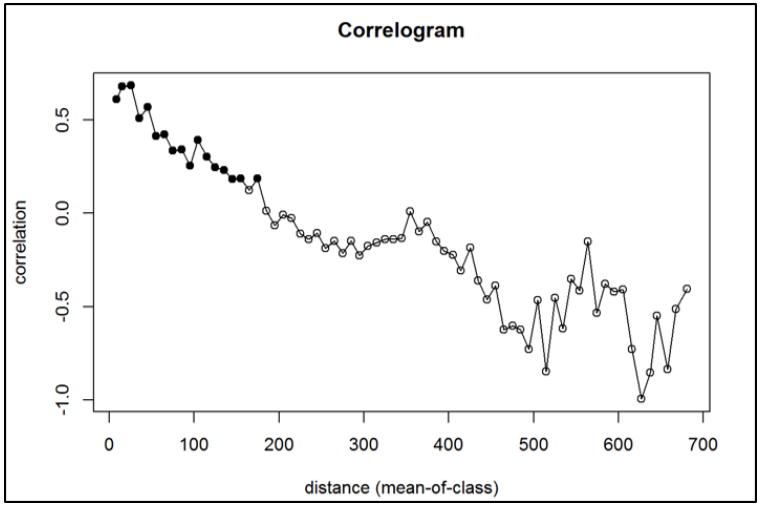
Igen, pozitív és erős térbeli autokorrelációt mértünk.

4. feladat (órai) - megoldás

```
alfold_elsa_800 <- elsa(x = alfold_felszinboritas, d =
 800, categorical = TRUE)
plot(alfold_elsa_800)
```



Korrelogram



Korrelogram

- a térbeli autokorreláltság távolságfüggésének bemutatására alkalmas diagram
- vízszintes tengely: távolságtartományok
- függőleges tengely: térbeli autokorreláció (Moran-féle I)
- teli (fekete) körök: szignifikáns pozitív autokorreláció
- jellemzően a közeli (kis távolságtartományhoz tartozó) pontok szignifikánsak

Korrelogram

`correlog(x, y, z, increment, resamp = 1000, na.rm = FALSE)`

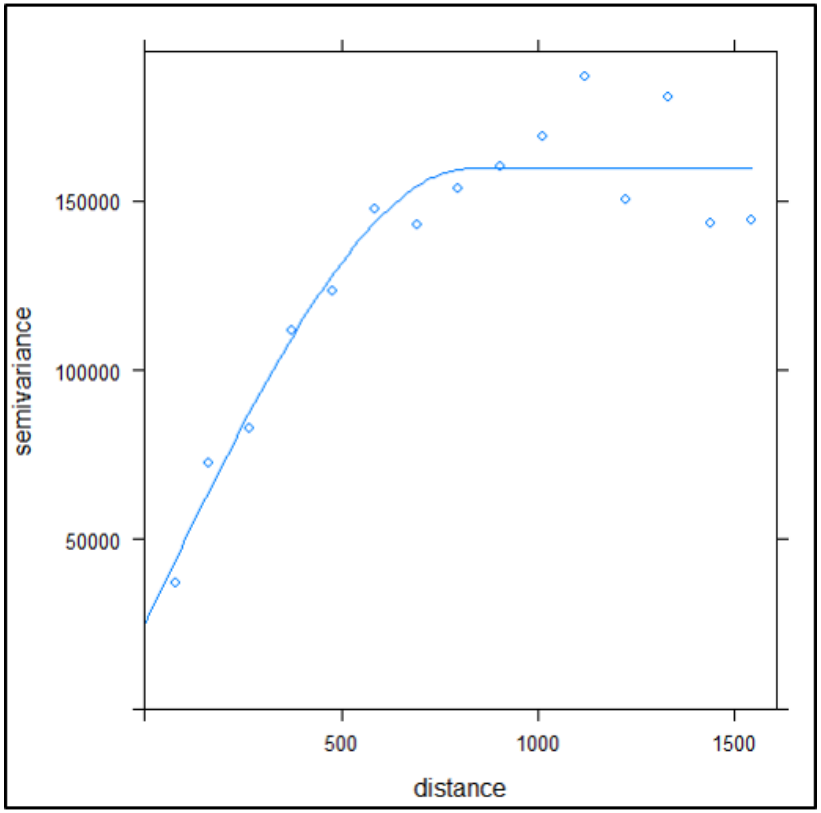
- az `ncf` csomag függvénye
- használható raszterekhez és vektorokhoz is
- szét kell szedni koordinátákra és adatokra, bármelyikkel is dolgozunk
- `x`, `y`: két koordinátavektor
- `z`: az értékeket tartalmazó vektor
- `increment`: távolságtartományok nagysága
- `resamp`: szignifikanciaszámításhoz hányszor vegyen mintát
- `na.rm`: elhagyja-e az ismeretlen értékeket (alapértelmezett: nem)

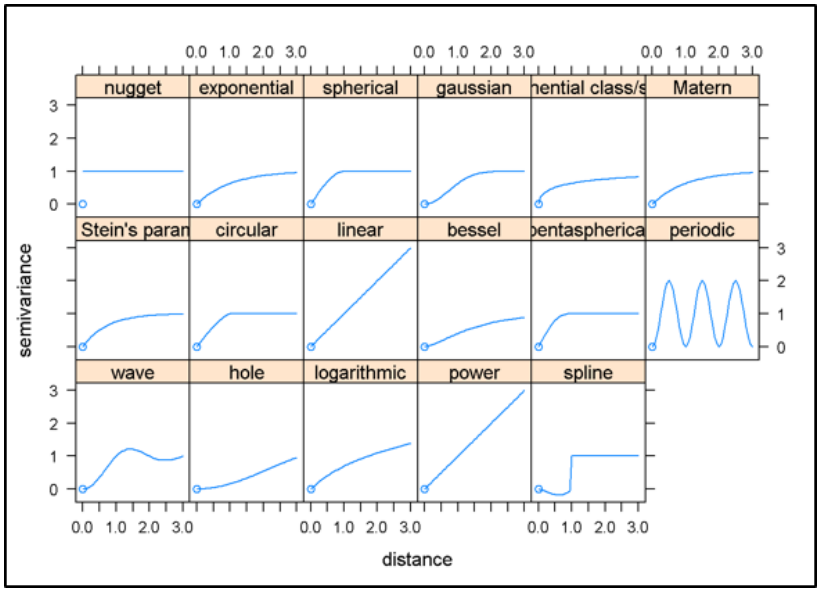
Korrelogram

Korrelogram rokonai

- kategorikus raszter esetén az `elsa` csomag `entrogram()` függvényével az ELSA-érték távolságfüggését ábrázolhatjuk
- a korrelogramhoz hasonló, de fordított alakú a félvariogram (semivariogram)
- a "tapasztalati félvariogramra" (= megfigyelt összefüggésre) félvariogrammodellt lehet illeszteni
- a félvariogrammodellel jellemezhető az adatok változatosságának távolságfüggése
- a krigelés (kriging) nevű interpolációs technika félvariogrammodellre támaszkodik
- a `gstat` geostatisztikai csomag függvényeivel lehet ezeket számolni

Félvariogram





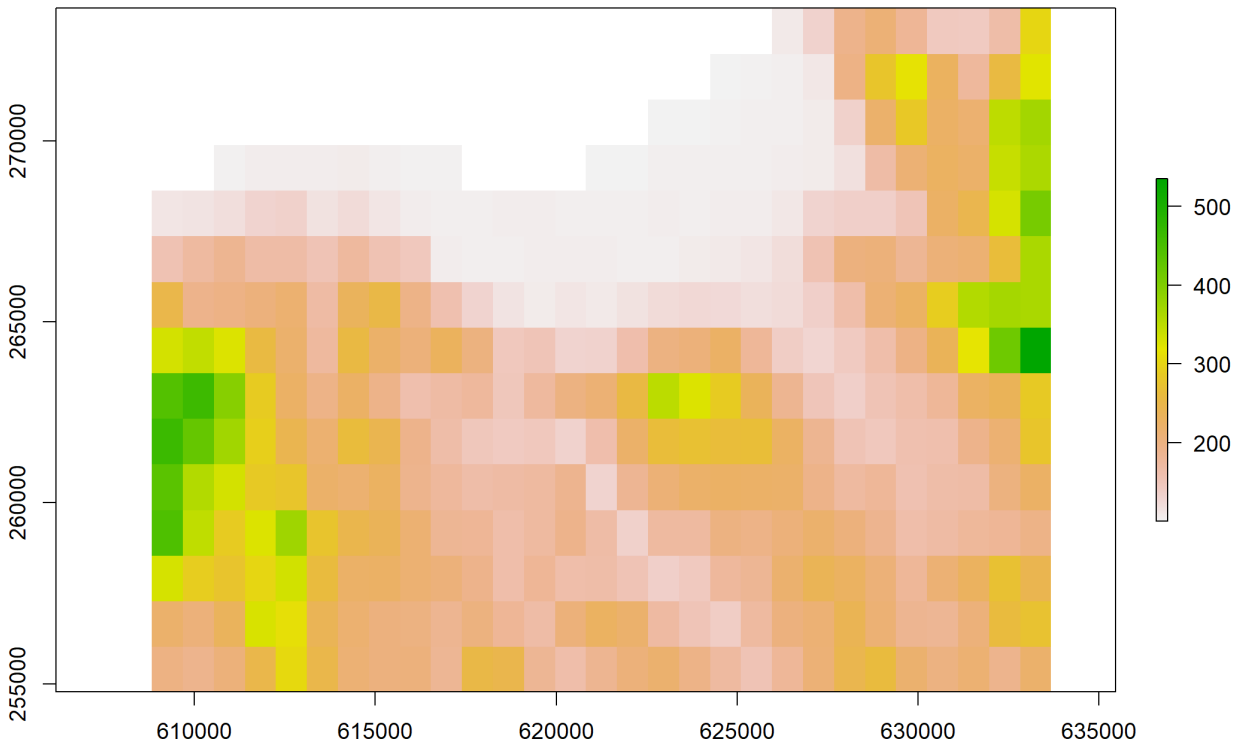
felfutó szakasz: van autokorreláltság

vízszintes szakasz: maximális variancia, nincs autokorreláltság

Korrelogram

Elég lassan számolható a korrelogram (és rokonai is), ezért lebutított adaton mutatom be.

```
esztergom_dem_10 <- aggregate(x = esztergom_dem, fact = 10)
plot(esztergom_dem_10)
```

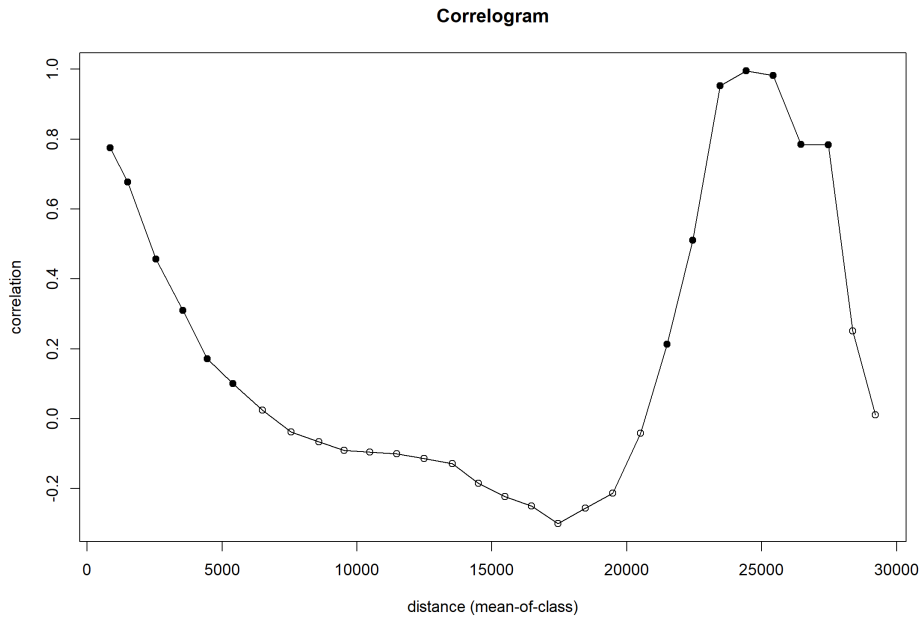


Korrelogram

Korrelogram számítása 1000 méteres távolságtarományokkal, 40 ismétléssel, ismeretlen értékeket elhagyva, raszterre:

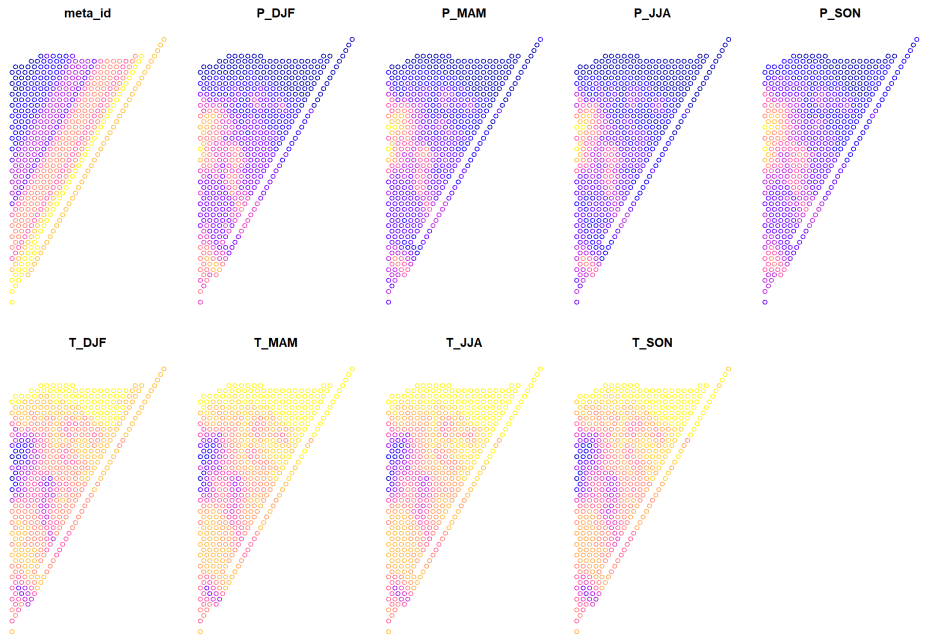
```
matrixkent <- rasterToPoints(x = esztergom_dem_10, spatial
 = FALSE)
```

```
plot(correlog(x = matrixkent[, 1], y = matrixkent[, 2], z
 = matrixkent[, 3], increment = 1000, resamp = 40, na.rm =
 TRUE))
```



Korrelogram

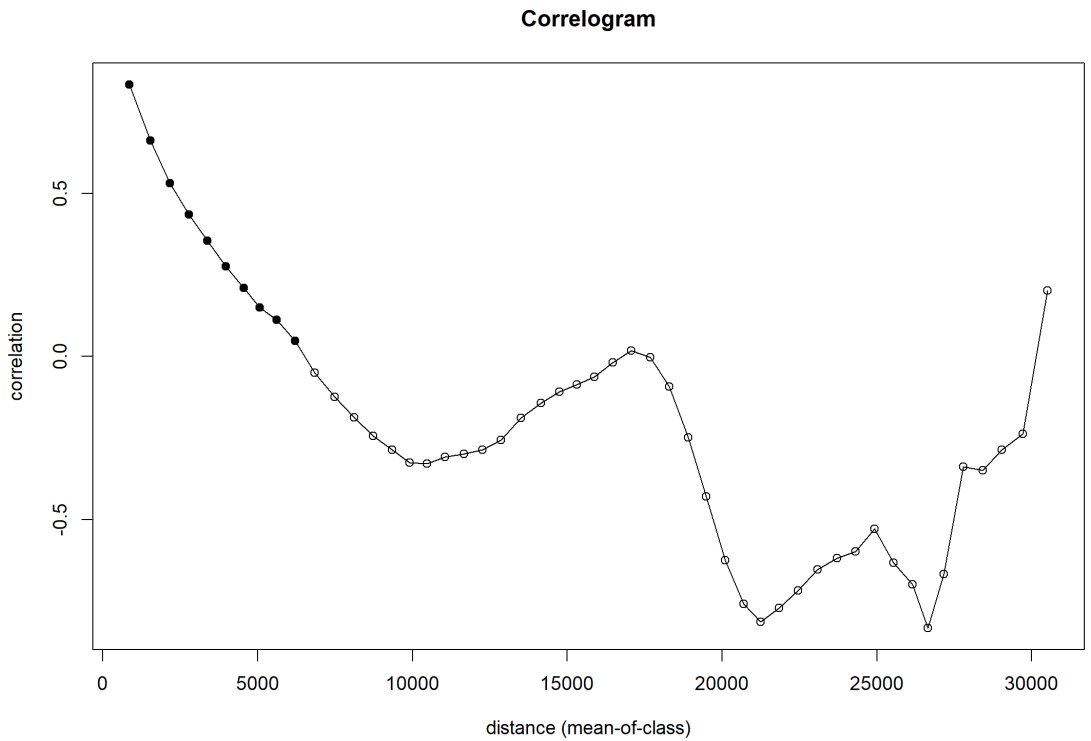
```
eghajlat_500 <- eghajlat[1:500, ]
plot(eghajlat_500)
```



Korrelogram

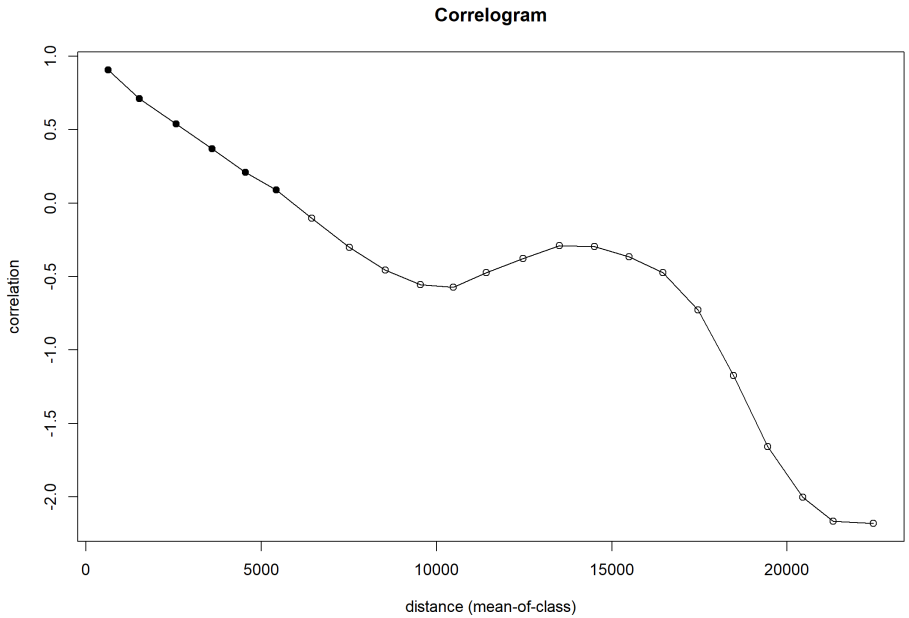
Korrelogram számítása 600 méteres távolságtarományokkal, 40 ismétléssel, ismeretlen értékeket megtartva, vektorra:

```
plot(correlog(x = st_coordinates(eghajlat_500)[, 1], y =
 st_coordinates(eghajlat_500)[, 2], z =
 eghajlat_500$T_MAM, increment = 600, resamp = 40))
```



5. feladat (órai)

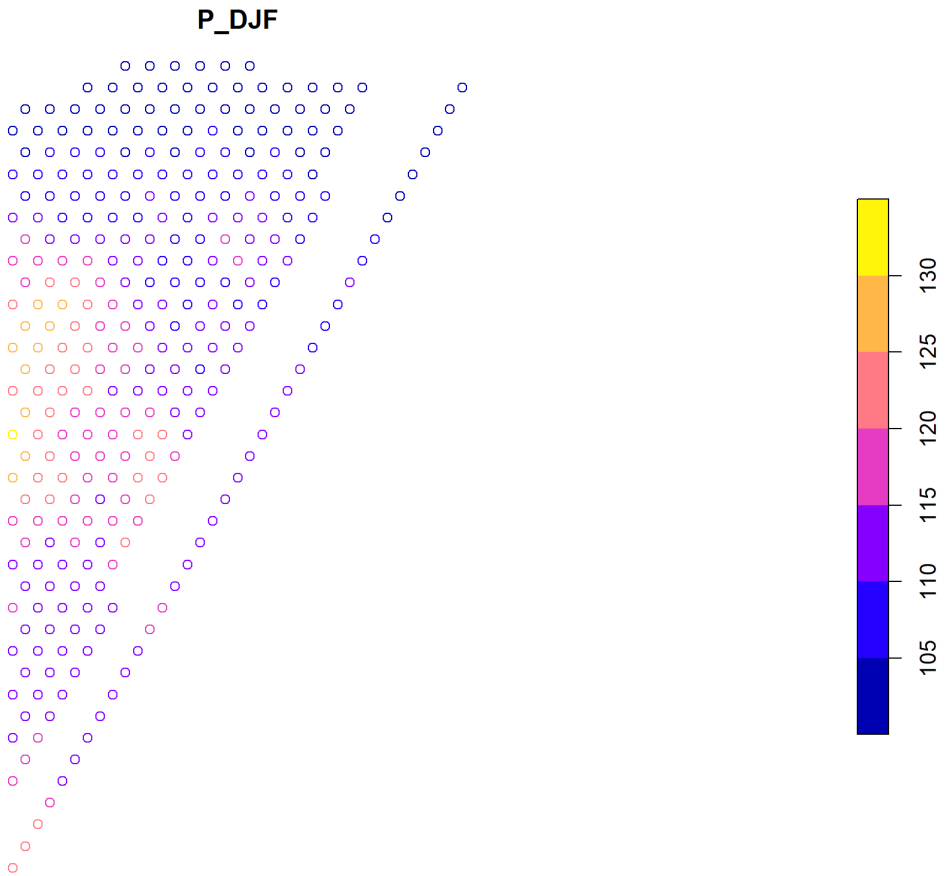
- Hozz létre az “eghajlat” nevű ponthalmaz első 300 eleméből egy leválogatást.
- Jelenítsd meg a kapott Simple Features-t a téli csapadék (“P_DJF”) alapján színezve.
- Számíts erre az éghajlati jellemzőre korrelogramot (és jelenítsd meg) 1000 méteres távolságtartományokat használva, 50 ismétlésben.



5. feladat (órai) - megoldás

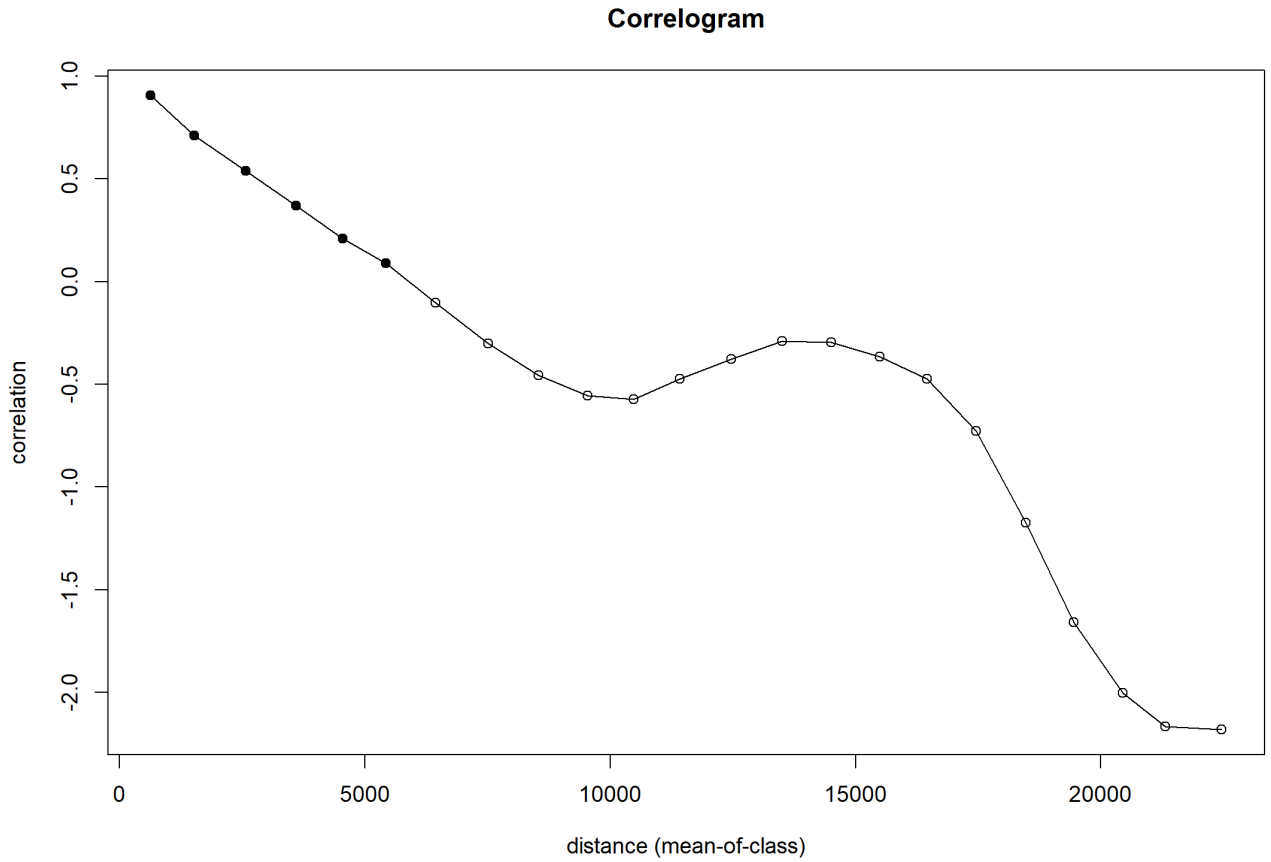
```
eghajlat_300 <- eghajlat[1:300, ]
```

```
plot(eghajlat_300[, "P_DJF"])
```



5. feladat (órai) - megoldás

```
plot(correlog(x = st_coordinates(eghajlat_300)[, 1], y =
 st_coordinates(eghajlat_300)[, 2], z =
 eghajlat_300$P_DJF, increment = 1000, resamp = 50))
```

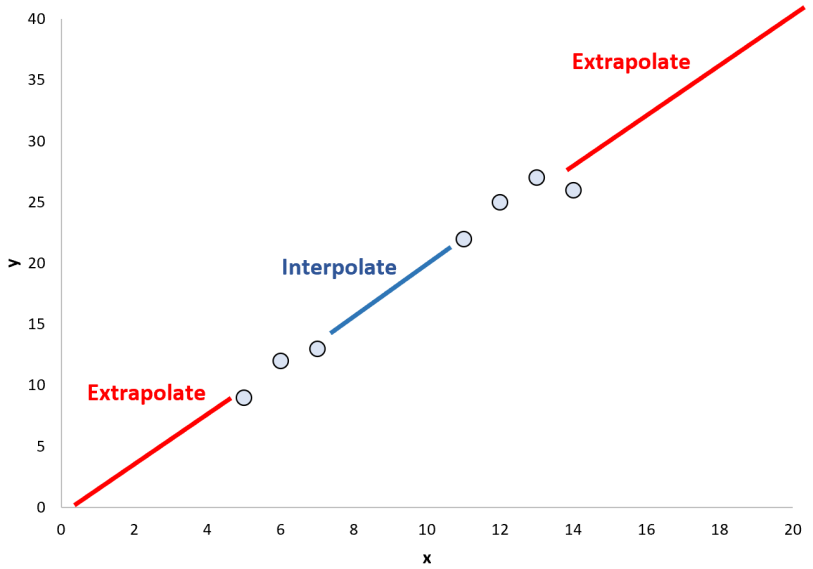


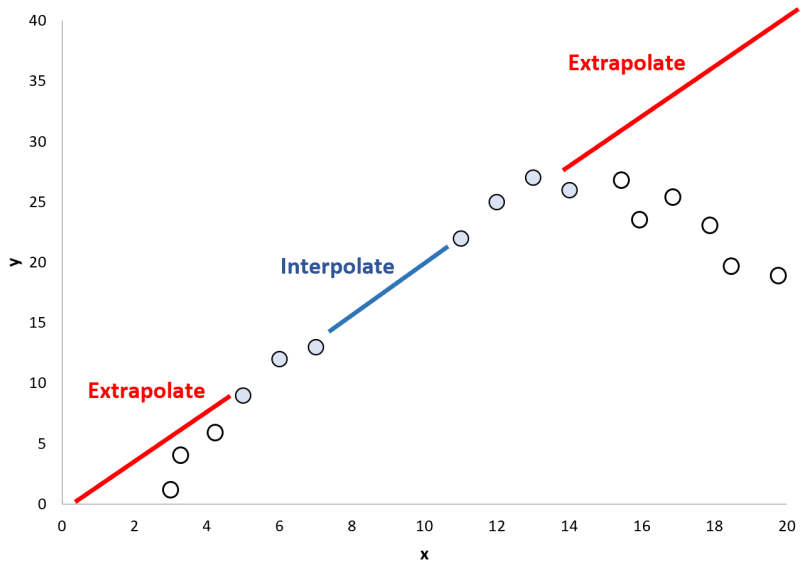
Section 4

Térbeli interpoláció

Térbeli interpoláció

- ismeretlen érték megbecslése ismert értékek segítségével
- interpoláció vs. extrapoláció
  - az extrapoláció mindig veszélyesebb
  - de valójában mindkettő becslés, mindkettő feltételezésekkel él
  - mögöttük mindig egy modell húzódik meg
  - legyen bár az egyszerű vagy összetett
- térbeli interpoláció
  - térbeli entitások (jellemzően pontok vagy rasztercellák) értékének megbecslése
  - a közeli vagy az összes ismert érték segítségével

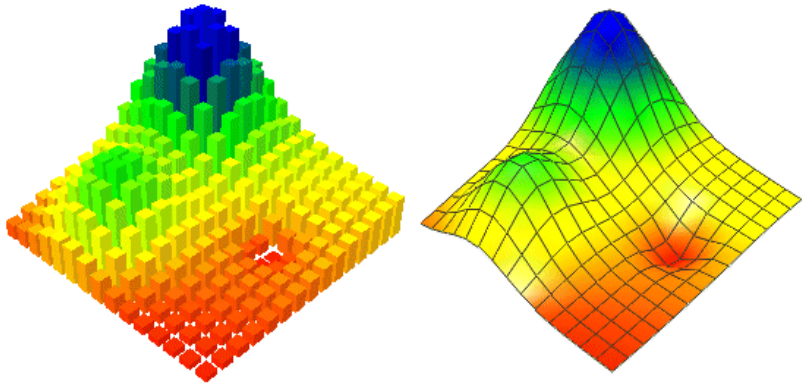


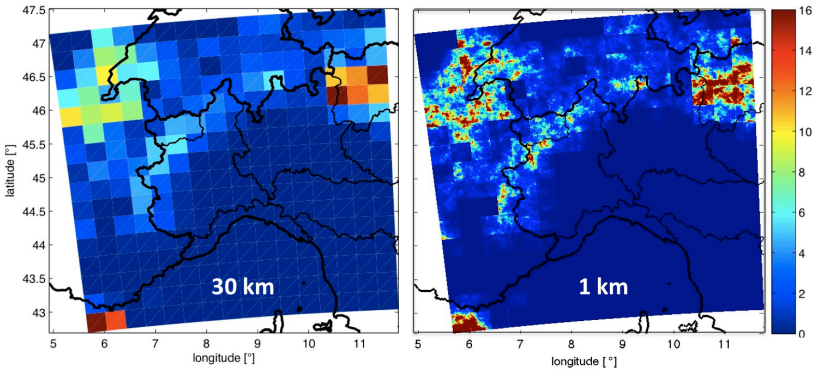


Térbeli interpoláció

Térbeli interpoláció lehetséges célja

- néhány ismeretlen pont értékének meghatározása
- adatok átvetítése új kiosztású pontrendszerbe (jellemzően szabályos elrendezésű pontrácsba)
- durva felbontásból finom felbontású (~folytonos) adatsor készítése
- utóbbit az éghajlatkutatásban statisztikai leskálázásnak (statistical downscaling) nevezik



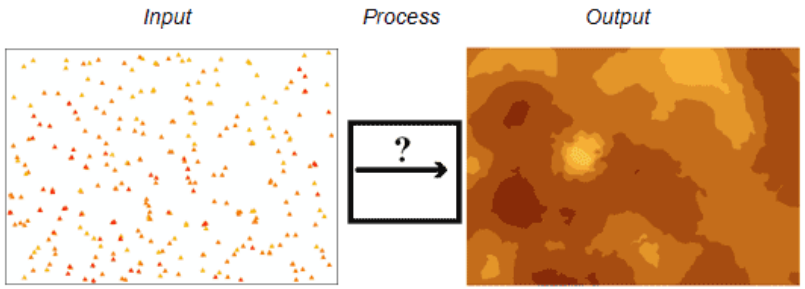


Térbeli interpoláció

Miért van rá szükség?

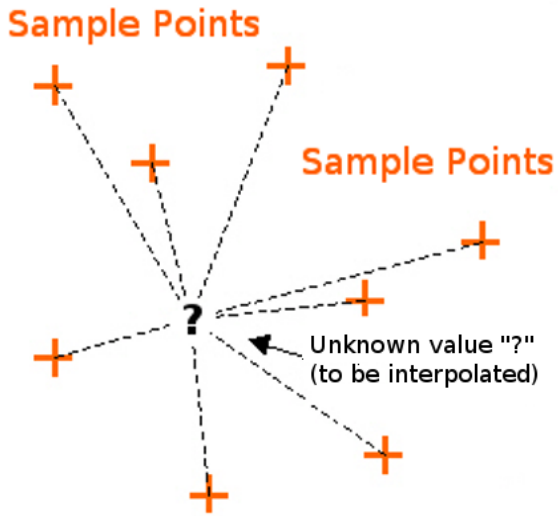
Mert az adatgyűjtés finom felbontásban

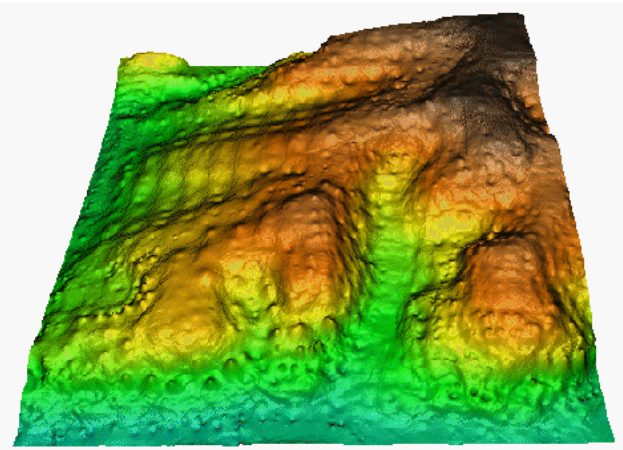
- időigényes,
- drága
- vagy lehetetlen.



Térbeli interpoláció

- interpolátor:
  - interpolációs modell/módszer
- sok interpolátor megadott számú pont értékeinek súlyozott átlagát számolja
  - legfőbb különbség a pontok számában és a súlyokban van
  - némelyik csak egy vagy néhány közeli pontot vesz figyelembe
  - más interpolátorok az összeset, de jellemzően a távolabbi pontokat kisebb súllyal





Térbeli interpoláció

Rengeteg interpolátor létezik.

- ezek közül csak néhányat mutatok be
- különböző R-csomagokból érhetőek el
- lásd a Geostatistics blokkot az “CRAN Task View: Analysis of Spatial Data” oldalon (link)
- az egyik legfontosabb csomag a `gstat`

Method/ package	ArcGIS/ ArcView GIS	GS+	R												S-PLUS	SURFER
			stats	akima	deldir	fields	geoR	geoRglm	GRASS	gstat	spatial	sgeostat	RandomFields	tripack		
NN	yes				yes									yes		yes
TIN	yes				yes									yes	yes	yes
NaN	yes															yes
Cl		yes	yes												yes	
TSA	yes										yes				yes	
IDW	yes	yes								yes						yes
LM	yes		yes												yes	yes
TPS	yes		yes	yes		yes									yes	
SK	yes					yes?	yes			yes		yes?	yes			
OK	yes	yes					yes		yes	yes			yes		yes	yes
UK	yes					yes	yes			yes	yes				yes	yes
SCK	yes									yes						
OCK	yes	yes								yes						
Universal CK	yes									yes						
BK	yes									yes					yes	
SCCK										yes						
KED							yes			yes						
StOK/StSK		yes								yes						
IK	yes									yes						
MBK							yes	yes								
Simulation										yes						

Térbeli interpoláció

Töltsük be a gstat csomagot, és készítsünk mintaadatot az interpolációhoz!
A felszínhőmérsékleti és magasságraszterekkel dolgozunk majd, de csökkentjük a felbontásukat a gyorsabb futás érdekében.

```
library(gstat)
```

```
magassag_raszter <- aggregate(x = domborzatmodell, fact =
 3, fun = mean, na.rm = FALSE)
homerseklet_raszter <- aggregate(x = felszinhomerseklet,
 fact = 3, fun = mean, na.rm = FALSE)
homerseklet_raszter <- mask(x = homerseklet_raszter, mask
 = magassag_raszter)
```

A domborzatmodell több ismeretlen cellát tartalmazott, ezeket átvisszük a másik raszterbe (maszkolunk).

Térbeli interpoláció

Hozzunk létre egy

- sűrű pontrácsot (valóság) és
- egy ritka, mintavételezett pontrácsot (megfigyelés)!

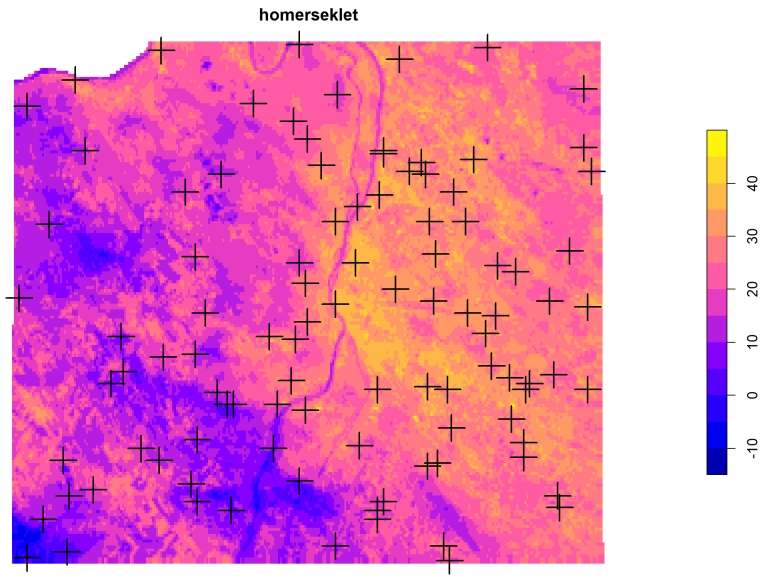
A ritka rács alapján fogjuk "tanítani" az interpolációs modellt, és predikciót adunk (=interpolálunk) a sűrű rács pontjaiba.
Remélhetőleg az interpolációnk jól fogja közelíteni majd a valóságot...

```
magassag_suru <- st_as_sf(rasterToPoints(x =
 magassag_raszter, spatial = TRUE))
homerseklet_suru <- st_as_sf(rasterToPoints(x =
 homerseklet_raszter, spatial = TRUE))
suru <- st_as_sf(cbind.data.frame(magassag_suru,
 st_drop_geometry(homerseklet_suru)))
colnames(suru)[colnames(suru) != attr(suru, "sf_column")]
 <- c("magassag", "homerseklet")
```

Térbeli interpoláció

A ritka rácsot véletlen mintavételezéssel készítjük a sűrű rácsból:

```
set.seed(12345)
kivalasztott_pontok_sorszama <- sample(x = 1:nrow(suru),
 size = 100, replace = FALSE)
ritka <- suru[kivalasztott_pontok_sorszama, ]
plot(suru[, "homerseklet"], pch = 15, reset = FALSE)
plot(st_geometry(ritka), pch = 3, cex = 3, lwd = 2, add =
 TRUE)
```



Távolsággal fordítottan arányos súlyozás

`idw(formula, locations, newdata, idp = 2)`

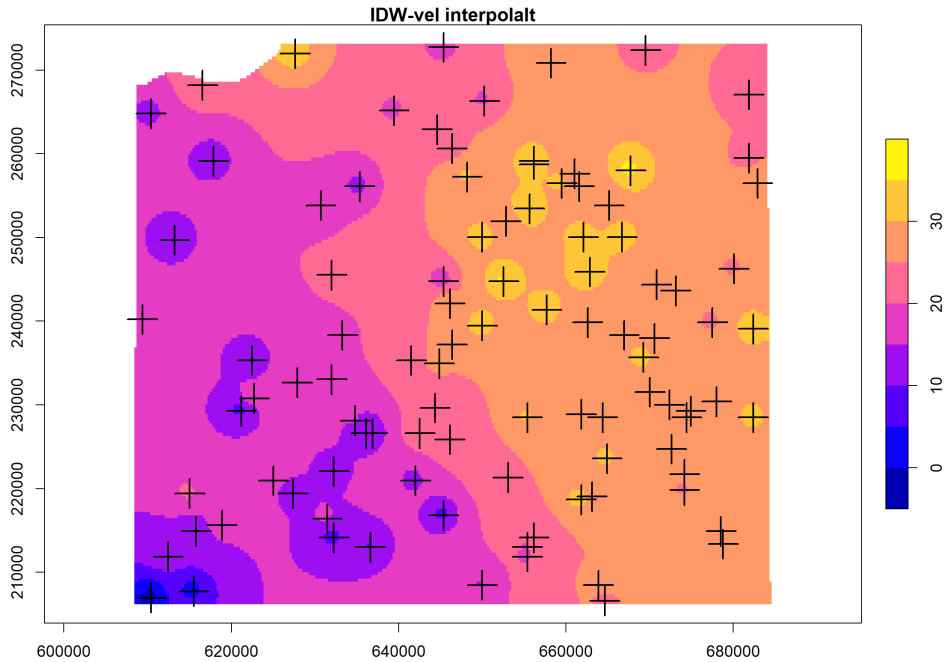
- távolsággal fordítottan arányos súlyozást (Inwerse Distance Weighting, IDW) végez
- `formula`: egy modellformula, mely idézőjelek nélkül tartalmazza az interpolálandó oszlop nevét, majd ezt: `~ 1`
- `locations`: pont típusú Simple Features, ez tartalmazza az interpolálandó oszlopot
- `newdata`: pont típusú Simple Features (nem kell, hogy legyen oszlopa), ezekre a helyekre fogunk interpolálni
- `idp`: a súlyozás hatványkitevője, alapértelmezetten 2
- az eredmény egy, a `newdata`-ra hasonlító pont típusú Simple Features 2 oszloppal
  - "var1.pred": ez tartalmazza az interpolált értéket
  - "var1.var": ez tartalmazza az interpoláláció varianciáját, de ez IDW esetén mindig ismeretlen

Távolsággal fordítottan arányos súlyozás

```
homerseklet_IDW <- idw(formula = homerseklet ~ 1,
 locations = ritka, newdata = suru, idp = 2)
```

```
[inverse distance weighted interpolation]
```

```
plot(homerseklet_IDW[, "var1.pred"], main = "IDW-vel
 interpolalt", pch = 15, axes = TRUE, reset = FALSE)
plot(st_geometry(ritka), pch = 3, cex = 3, lwd = 2, add =
 TRUE)
```



Krigelés

A krigelésnek nagyon sok típusa van, most csak kettőt nézünk meg:

- szokásos/szokványos krigelés (Ordinary Kriging, OK)
- regressziókrigelés/regressziós krigelés (Regression Kriging, RK)

Az előbbi az IDW-hez hasonlóan nem használ segédváltozót, míg az utóbbi igen.

Az IDW determinisztikus interpolátor, míg a krigelés sztochasztikus/geostatisztikai interpolátor.

- Vagyis a krigeléshez előbb a interpolálandó változó térbeli struktúráját fel kell tárnunk.
- Ehhez félvariogrammodellt fogunk készíteni.

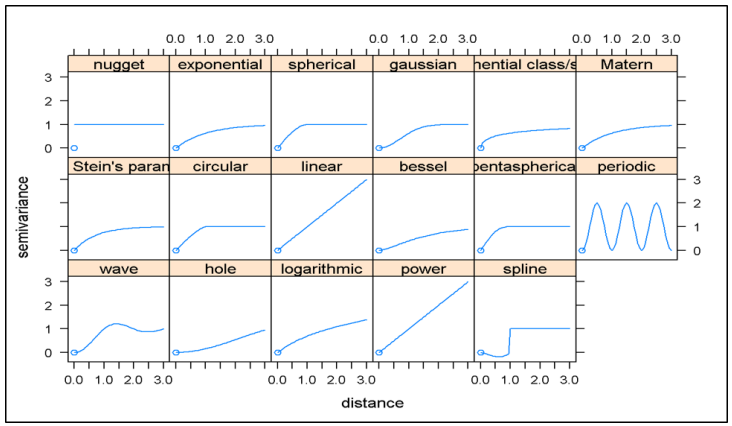
Félvariogram

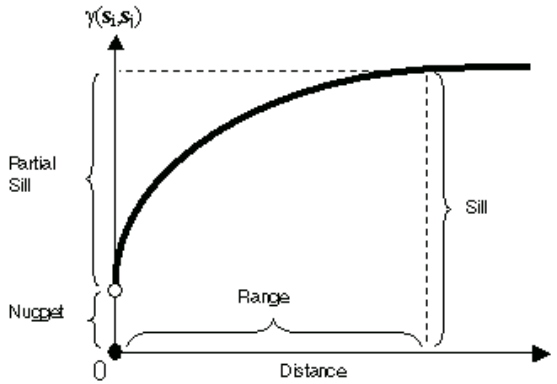
Sokféle (fél)variogrammodell létezik, pl.:

- exponenciális
- gauss-i
- szférikus

A modellek a típuson kívül három jellemzővel írhatóak le:

- nugget (röghatás)
- sill (küszöbszint)
- *e kettő különbsége a partial sill*
- range (hatástávolság)





Félvariogram

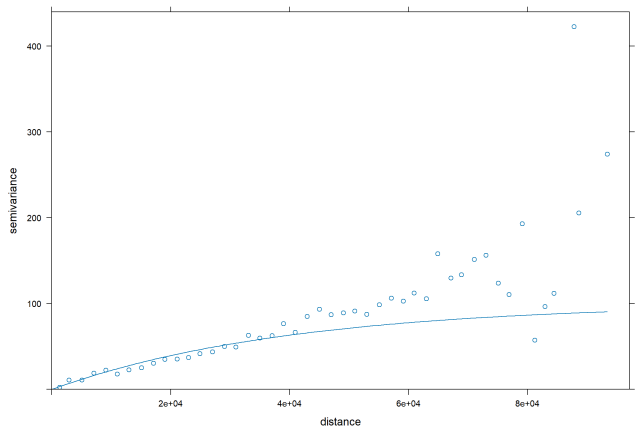
A variogrammodell készítésének három lépése:

- tapasztalati variogramfelhő számítása/ábrázolása (`variogram()`)
- kezdeti variogrammodell létrehozása kézzel megadott értékekkel (`vgm()`)
- kezdeti variogrammodell automatikus illesztése a tapasztalati variogramfelhőhöz (`fit.variogram()`)

Félvariogram

Ábrázoljuk a tapasztalati variogramfelhőt 2 km-es osztásokkal, 100 km-es távolságig, és rakjunk rá egy exponenciális, röghatás nélküli variogrammodellt 100-as értékű, részleges küszöbszinttel és 40 km-es hatástávolsággal!

```
tapasztalati_variogramfelho <- variogram(object =
 homerseklet ~ 1, locations = ritka, cutoff = 100000,
 width = 2000)
kezdeti_variogrammodell <- vgm(model = "Exp", nugget = 0,
 psill = 100, range = 40000)
plot(tapasztalati_variogramfelho, kezdeti_variogrammodell)
```



Félvariogram

`variogram(object, locations, cutoff, width)`

- tapasztalati variogramfelhőt számol
- `object`: egy modellformula, megegyezik a krigelésnél használt modellformulával (később...)
- `locations`: pont típusú Simple Features, ez tartalmazza a modellformulában megjelölt oszlopo(ka)t
- `cutoff`: maximális vizsgálandó távolság
- `width`: osztások távolsága

`vgm(psill, model, nugget, range)`

- variogrammodellt készít kézi beállításokkal
- `psill`: röghatás nélküli küszöbszint (partial sill)
- `model`: modelltípus (pl. “Exp”, “Gau”, “Sph”)
- `nugget`: röghatás
- `range`: hatástávolság

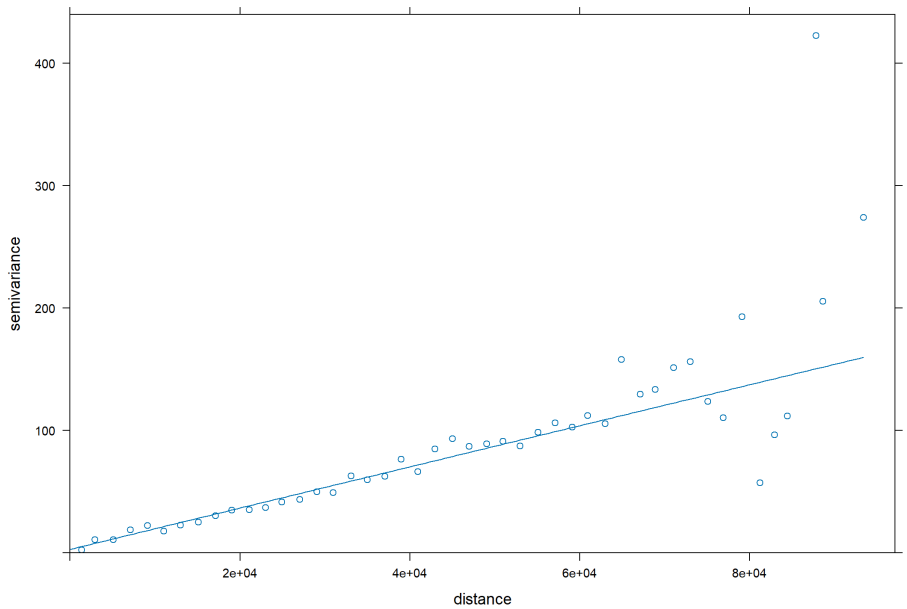
Félvariogram

`fit.variogram(object, model)`

- kezdeti variogrammodellt illeszt automatikusan a tapasztalati variogramfelhőhöz
- `object`: tapasztalati variogramfelhő, amit a `variogram()` függvénnyel számoltunk
- `model`: kezdeti variogrammodell, amit a `vgm()` függvénnyel hoztunk létre
- alapértelmezetten mindhárom paramétert (`nugget`, `psill`, `range`) illeszti, de ezt felülbírálhatjuk a `fit.sills` és `fit.ranges` opcionális logikai paraméterekkel

Félvariogram

```
illesztett_variogrammodell <- fit.variogram(object =
 tapasztalati_variogramfelho, model =
 kezdeti_variogrammodell)
plot(tapasztalati_variogramfelho,
 illesztett_variogrammodell)
```



Ez már szépen illeszkedik, ez a variogrammodell jó lesz a krigeléshez...

Krigelés

`krige(formula, locations, newdata, model)`

- `formula`: egy modellformula, mely idézőjelek nélkül tartalmazza bal oldalon az interpolálandó oszlop nevét, majd a hullámvonal után vagy az 1-es számot (szokványos krigelés), vagy a segédváltozó(k) nevét (regressziós krigelés)
- `locations`: pont típusú Simple Features, ez tartalmazza a modellformulában megjelölt oszlopo(ka)t
- `newdata`: pont típusú Simple Features, ezekre a helyekre fogunk interpolálni, tartalmazza a segédváltozókat regressziós krigelés esetén
- `model`: az illesztett variogrammodell, amely leírja az adataink térbeli struktúráját
- az eredmény egy, a `newdata`-ra hasonlító pont típusú Simple Features 2 oszloppal
  - "var1.pred": ez tartalmazza az interpolált értéket
  - "var1.var": ez tartalmazza az interpoláláció varianciáját

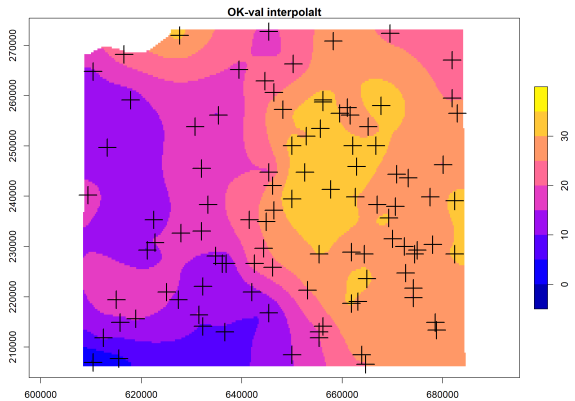
Krigelés

Szokványos krigelés (nem használunk segédváltozót):

```
homerseklet_OK <- krige(formula = homerseklet ~ 1,
 locations = ritka, newdata = suru, model =
 illesztett_variogrammodell)
```

```
[using ordinary kriging]
```

```
plot(homerseklet_OK[, "var1.pred"], main = "OK-val
 interpolalt", pch = 15, axes = TRUE, reset = FALSE)
plot(st_geometry(ritka), pch = 3, cex = 3, lwd = 2, add =
 TRUE)
```



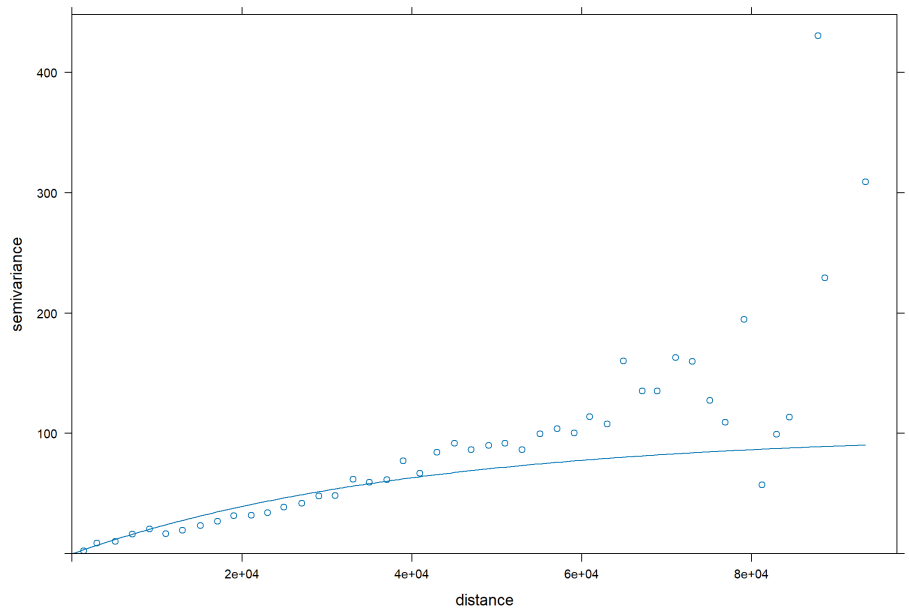
Krigelés

Regressziós krigelés (segédváltozó gyanánt a tszf. magasságot használjuk):

- minden ugyanúgy történik, mint a szokványos krigelésnél
- csak a variogram() és krige() függvényekben haszált formulát lecseréljük: "~ 1" $\rightarrow$ "$\sim$ magassag"
- ha több háttérváltozónk lenne, akkor így: "valaszvaltozo $\sim$ hattervaltozo1 + hattervaltozo2 + hattervaltozo3"
- *ha háttérváltozóként a két koordinátát használjuk, akkor univerzális krigelésnek (Universal Kriging, UK) nevezzük a módszert, de a lényege ugyanaz*
- a háttérváltozó(ka)t nem csak a ritka pontrácsnak kell tartalmaznia (amit az interpolációs modell megtanul), hanem a sűrű pontrácsnak is (ami alapján az interpolációs modellel predikálunk)

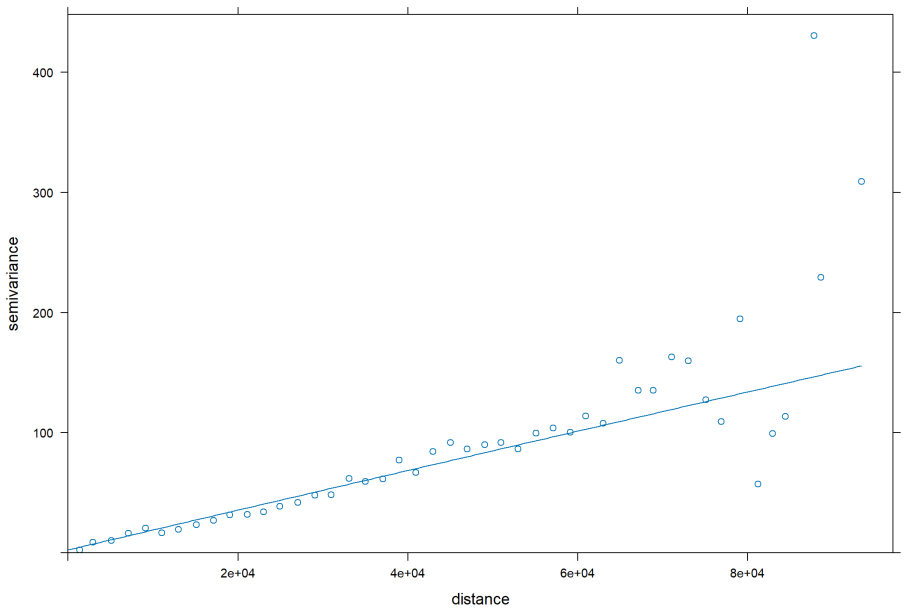
Krigelés

```
tapasztalati_variogramfelho <- variogram(object =
 homerseklet ~ magassag, locations = ritka, cutoff =
 100000, width = 2000)
kezdeti_variogrammodell <- vgm(model = "Exp", nugget = 0,
 psill = 100, range = 40000)
plot(tapasztalati_variogramfelho, kezdeti_variogrammodell)
```



Krigelés

```
illesztett_variogrammodell <- fit.variogram(object =
 tapasztalati_variogramfelho, model =
 kezdeti_variogrammodell)
plot(tapasztalati_variogramfelho,
 illesztett_variogrammodell)
```



Krigelés

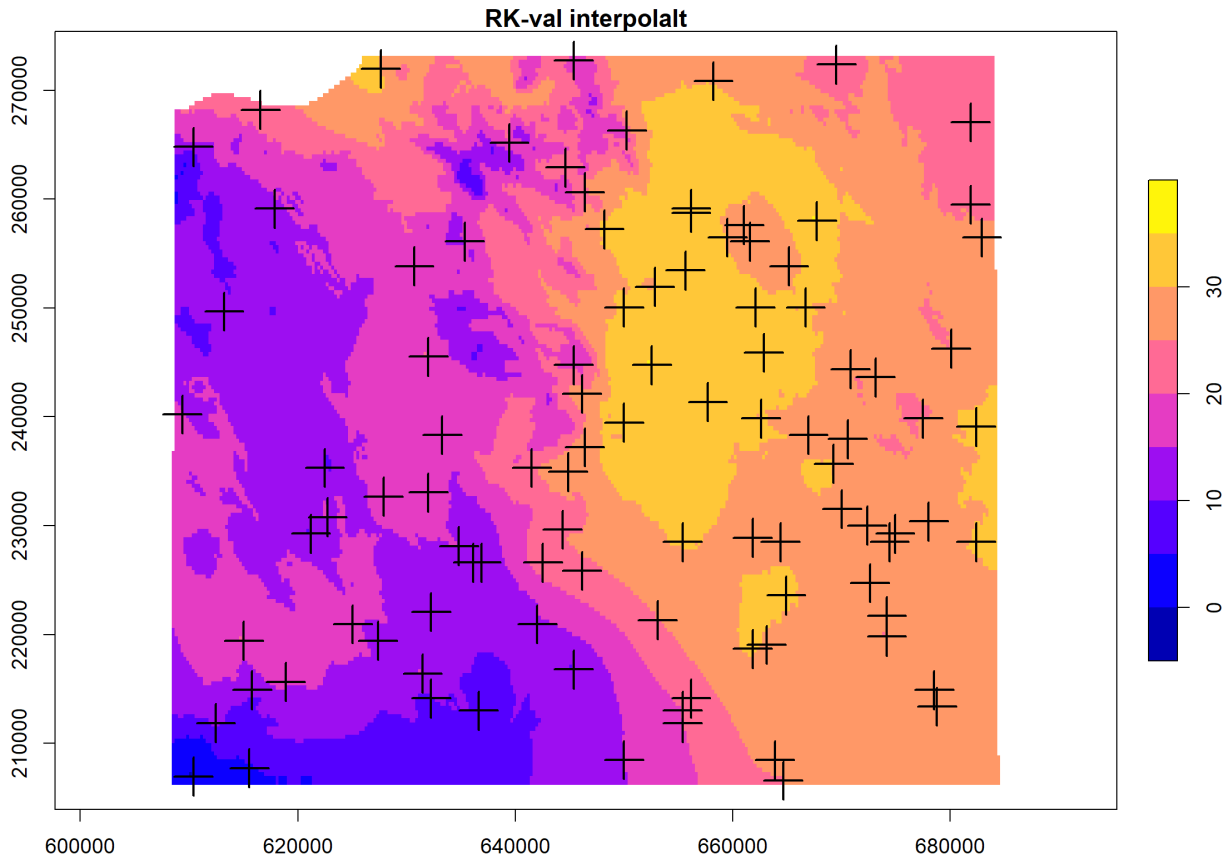
Regressziós krigelés (segédváltozó gyanánt a tszf. magasságot használjuk):

```
homerseklet_RK <- krige(formula = homerseklet ~ magassag,
 locations = ritka, newdata = suru, model =
 illesztett_variogrammodell)
```

```
[using universal kriging]
```

Krigelés

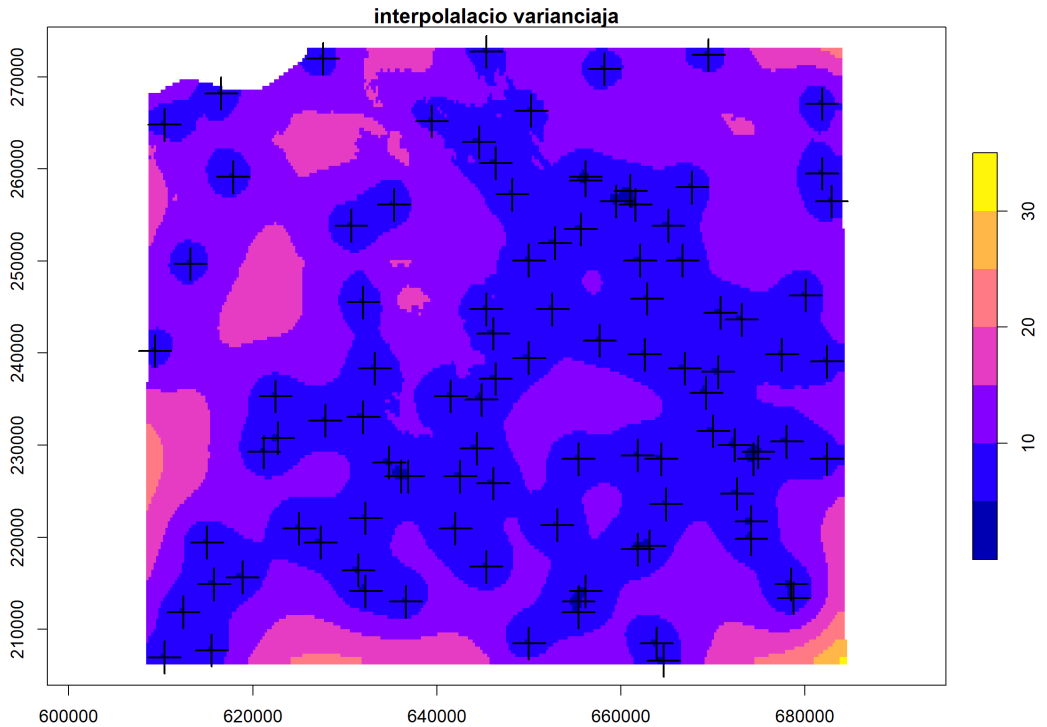
```
plot(homerseklet_RK[, "var1.pred"], main = "RK-val
 interpolalt", pch = 15, axes = TRUE, reset = FALSE)
plot(st_geometry(ritka), pch = 3, cex = 3, lwd = 2, add =
 TRUE)
```



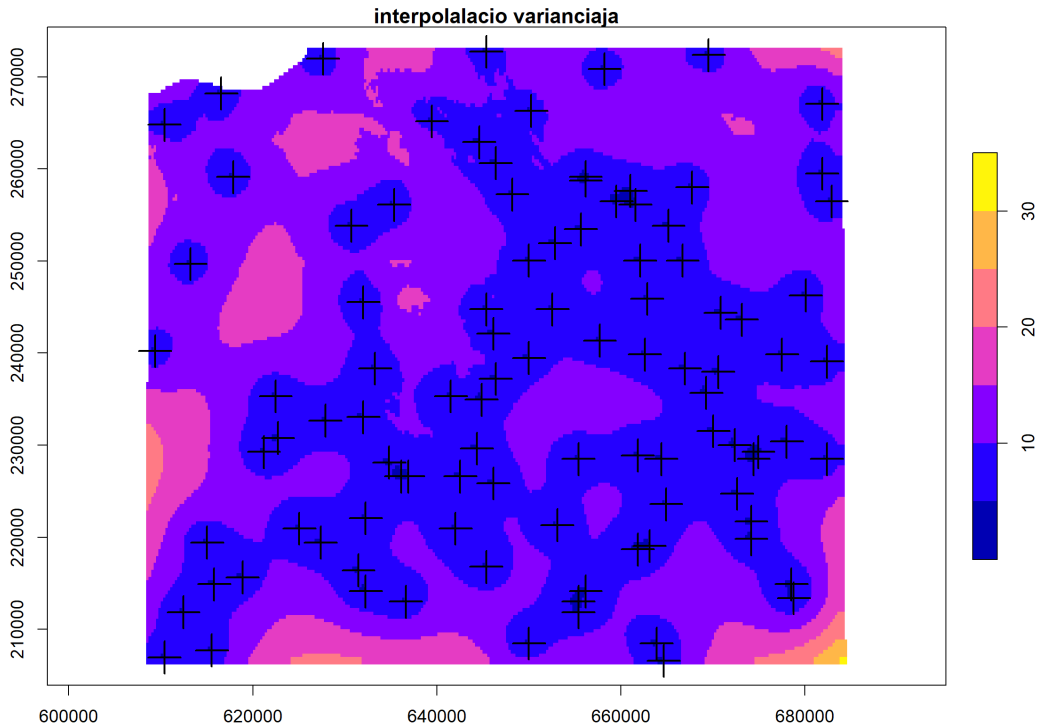
Krigelés

Nézzük meg az interpoláció varianciáját is:

```
plot(homerseklet_RK[, "var1.var"], main = "interpolalacio
 varianciaja", pch = 15, axes = TRUE, reset = FALSE)
plot(st_geometry(ritka), pch = 3, cex = 3, lwd = 2, add =
 TRUE)
```



Krigelés



A modell tanításához használt pontok közelében kicsi, távolabb nagy.
Ez nem túl hasznos infó. Sokkal praktikusabb

- a valósággal összehasonlítani (ha ismert, lásd a 6. feladatban), vagy
- keresztvalidációt alkalmazni (Leave-one-out Cross-validation, LOOCV), lásd a `krige.cv()` függvényt.

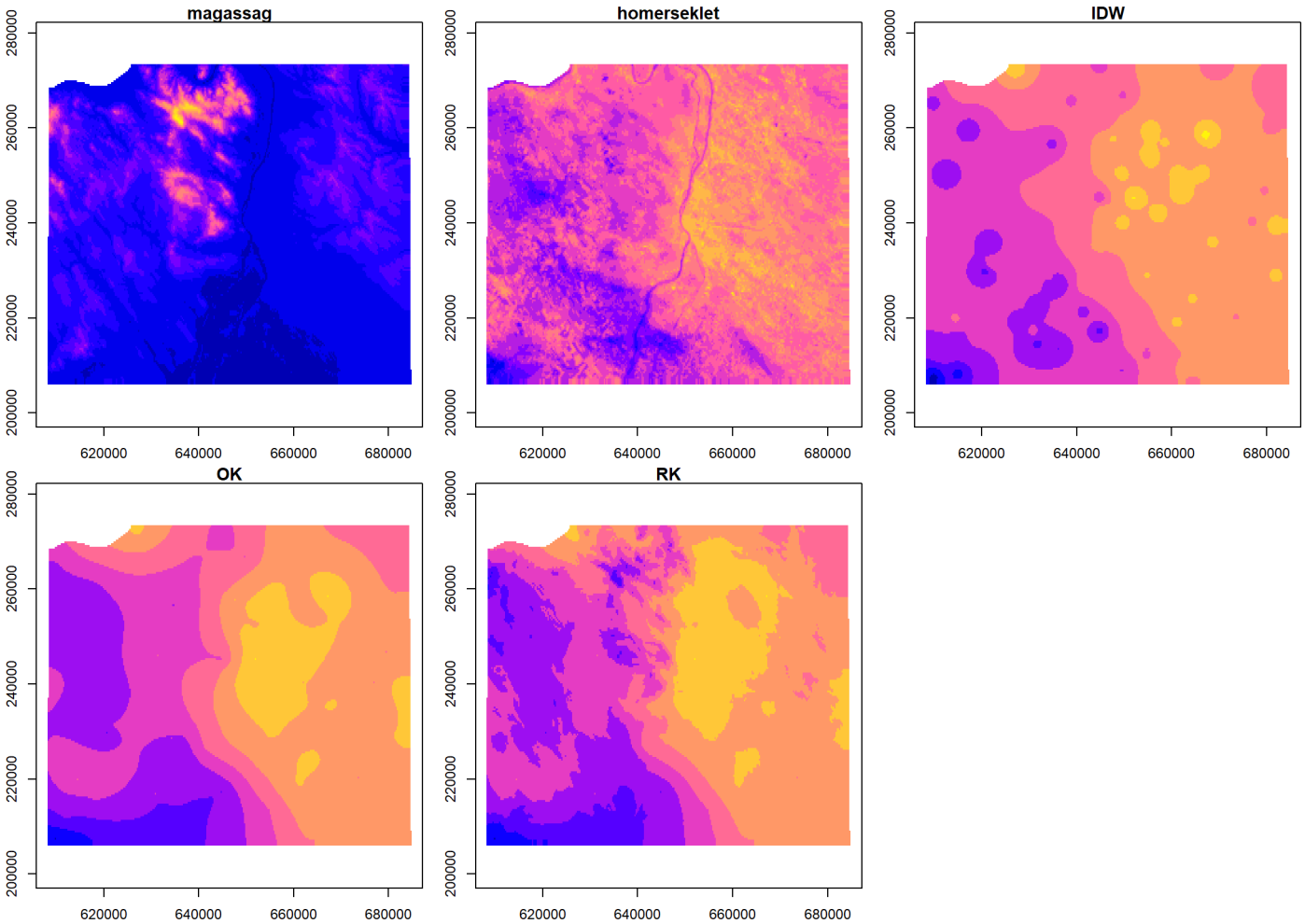
Krigelés

Hasonlítsuk össze a három interpolációnkat:

```
suru$IDW <- homerseklet_IDW$var1.pred
suru$OK <- homerseklet_OK$var1.pred
suru$RK <- homerseklet_RK$var1.pred
```

```
plot(suru, pch = 15, axes = TRUE, reset = FALSE)
plot(st_geometry(ritka), pch = 3, cex = 3, lwd = 2, add =
 TRUE)
```

Krigelés



A regressziós krigelés adta messze a legértelmesebb eredményt.

6. feladat (házi)

- Rögzítsd a véletlenszámgenerátort egy neked tetsző értéken.
- Válogass le a sűrű pontrácsból 300 darab, véletlen módon kiválasztott pontot.
- Interpoláld e 300 pontban mért magasságértékeket a sűrű rács pontjaiba távolsággal fordítottan arányos súlyozással, 1,5-ös kitevőt használva.
- Fűzd össze egy Simple Features-zé a sűrű rács eredeti magasságértékeit és az interpolált magasságokat.
- Nevezd el beszédes néven e két oszlopot.
- Harmadik oszlopként fűzd hozzájuk az interpolált és az eredeti érték különbségét.
- Jelenítsd meg e három oszlopot egymás mellett, négyzetes (15-ös) pontjelet alkalmazva, tengelyfeliratokkal és koordinátaráccsal.

6. feladat (házi) - megoldás

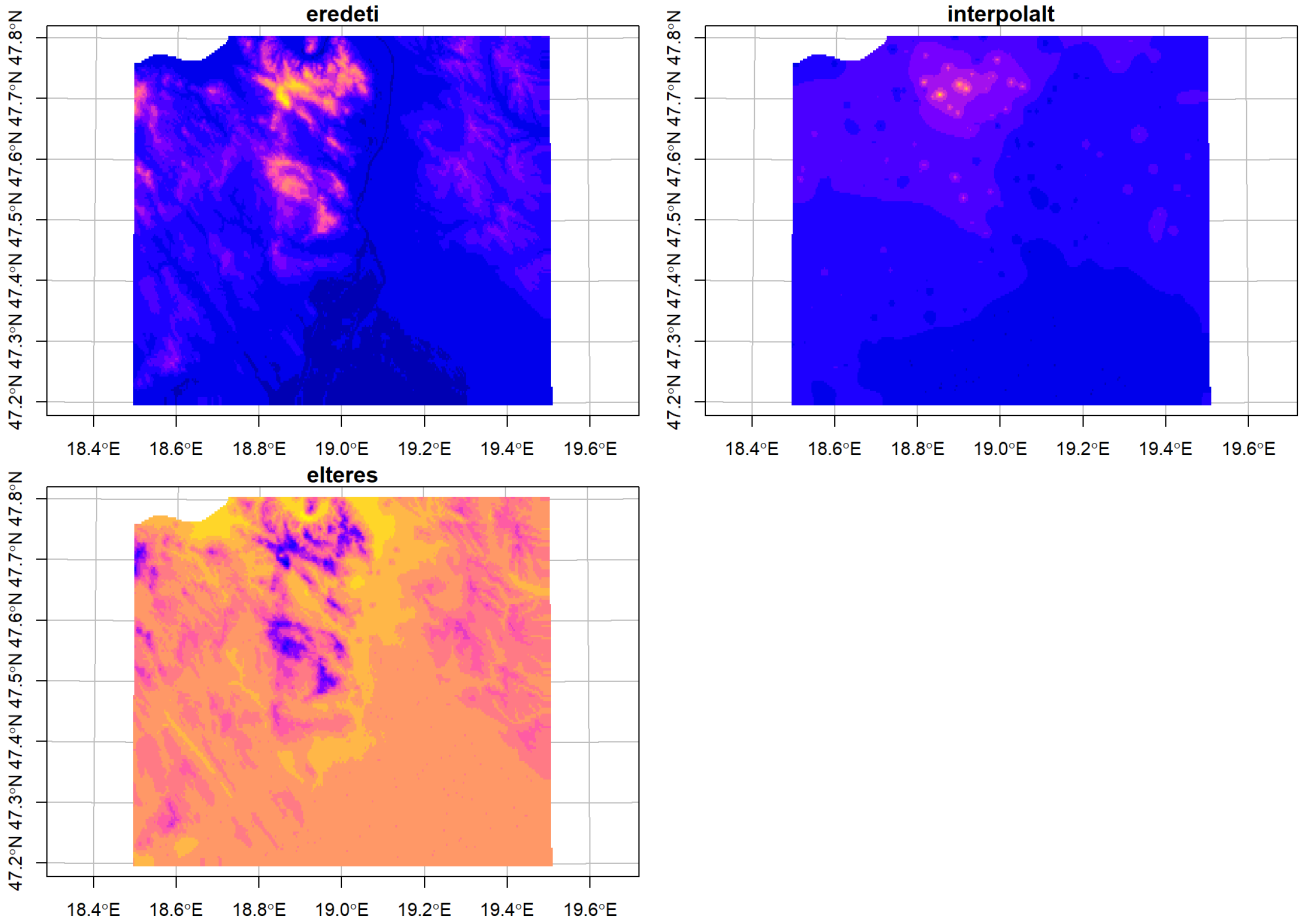
```
set.seed(10101)
pontok_300 <- sample(x = 1:nrow(suru), size = 300, replace
 = FALSE)
ritka_300 <- suru[pontok_300, ]
magassag_IDW <- idw(formula = magassag ~ 1, locations =
 ritka_300, newdata = suru, idp = 1.5)
```

[inverse distance weighted interpolation]

```
eredeti_es_interpolalt <- st_as_sf(cbind.data.frame(suru[,
 "magassag"], st_drop_geometry(magassag_IDW[,
 "var1.pred"])))
colnames(eredeti_es_interpolalt)[colnames(eredeti_es_interpolalt)
 != attr(eredeti_es_interpolalt, "sf_column")] <-
 c("eredeti", "interpolalt")
eredeti_es_interpolalt$elteres <-
 eredeti_es_interpolalt$interpolalt -
 eredeti_es_interpolalt$eredeti
```

6. feladat (házi) - megoldás

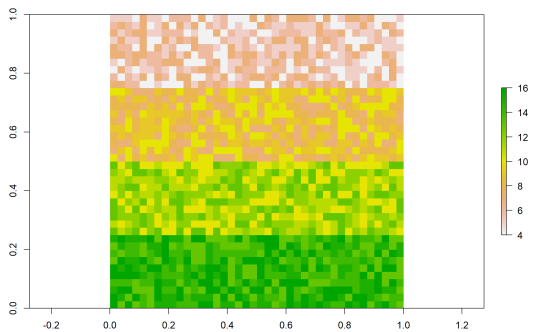
```
plot(eredeti_es_interpolalt, pch = 15, graticule = TRUE,
 axes = TRUE)
```



7. (összefoglaló) feladat (házi)

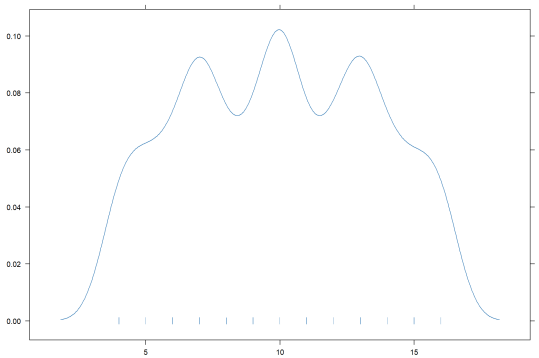
- Rögzítsd a véletlenszámgenerátort egy neked tetsző értéken.
- Hozzál létre két 40×40-es rasztert, melyek az 1, 2, 3, és 4 számokat tartalmazzák:
  - az egyik (“raszter_veletlen”) véletlen kiosztásban,
  - a másik (“raszter_blokkos”) a függőlegesen négy negyedbe kiosztva a számokat
- Képezz egy harmadik rasztert is (“raszter_osszeg”), amely minden cellájában a blokkos raszter adott cellája háromszorosának és a véletlen raszter adott cellaértékének összegét tartalmazza.
- Jelenítsd meg ezt a rasztert.

...



7. (összefoglaló) feladat (házi)

- Számold ki és jelenítsd meg a véletlen raszter entrópiaalapú térbeli függőségét úgy,
  - hogy a cellaértékeket kategorikusnak veszed,
  - a vizsgálati távolság pedig legyen a raszter vízszintes irányú felbontásának ötszöröse.
- Számold ki a blokkos raszter globális Moran-féle térbeli autokorrelációját a 3×3-as bástya súlymátrixot használva.
- Jelenítsd meg az összegzett raszter lokális Moran-féle térbeli autokorrelációját ugyanevvel a súlymátrixszal.
- Jelenítsd meg az összegzett raszter cellaértékeinek gyakoriségi eloszlását diagramon.

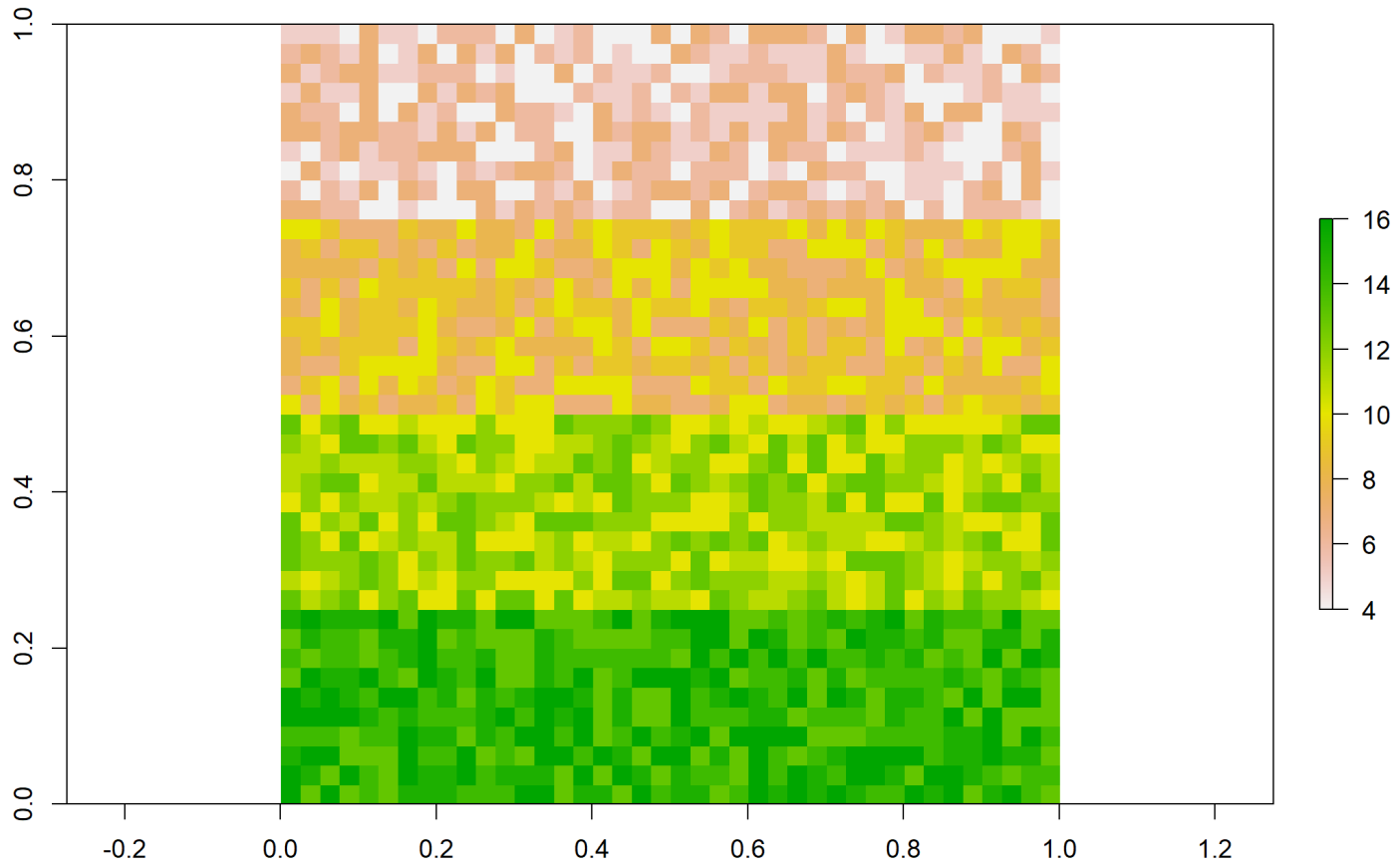


7. (összefoglaló) feladat (házi) - megoldás

```
set.seed(12345)
szamok_veletlen <- sample(x = 1:4, size = 40 * 40, replace
 = TRUE)
matrix_veletlen <- matrix(data = szamok_veletlen, ncol =
 40, nrow = 40)
raszter_veletlen <- raster(x = matrix_veletlen)
szamok_blokkos <- rep(x = 1:4, each = 40 * 10)
matrix_blokkos <- matrix(data = szamok_blokkos, ncol = 40,
 nrow = 40, byrow = TRUE)
raszter_blokkos <- raster(x = matrix_blokkos)
raszter_osszeg <- raszter_veletlen + raszter_blokkos * 3
```

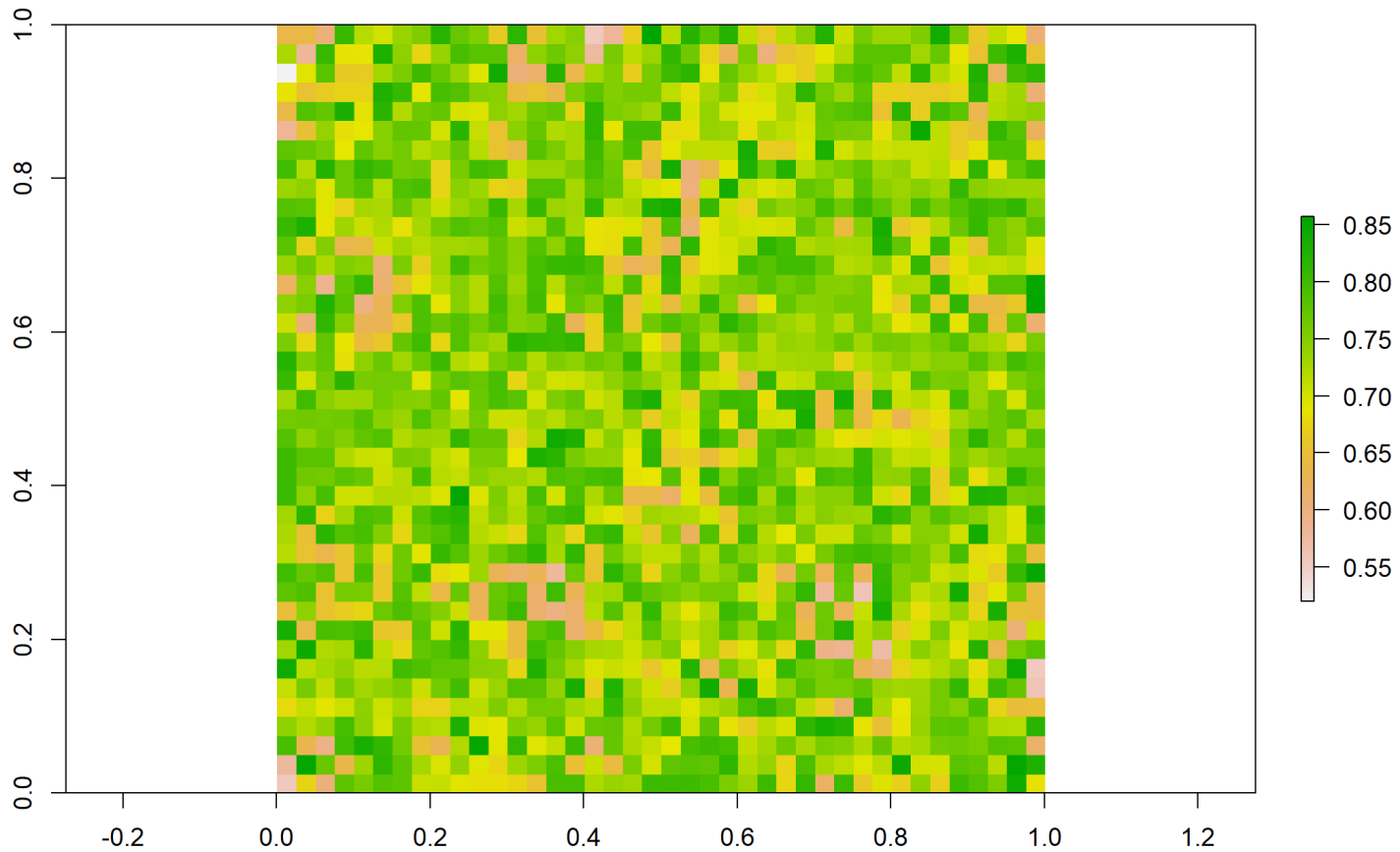
7. (összefoglaló) feladat (házi) - megoldás

```
plot(raszter_osszeg)
```



7. (összefoglaló) feladat (házi) - megoldás

```
plot(elsa(x = raszter_veletlen, d = xres(raszter_veletlen)
 * 5, categorical = TRUE))
```

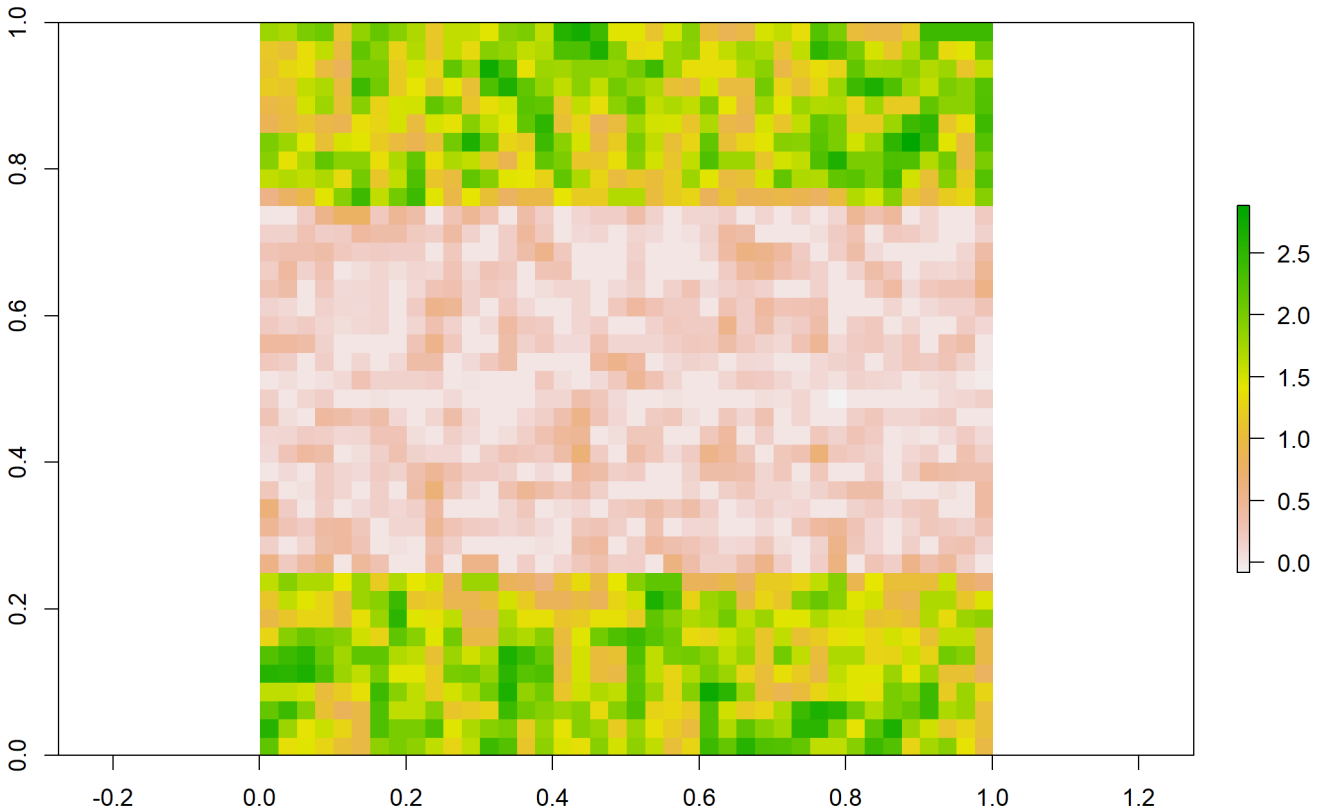


7. (összefoglaló) feladat (házi) - megoldás

```
Moran(x = raszter_blokkos, w = bastya3)
```

```
[1] 0.974359
```

```
plot(MoranLocal(x = raszter_osszeg, w = bastya3))
```



7. (összefoglaló) feladat (házi) - megoldás

```
densityplot(raszter_osszeg)
```