Package 'lineaDEM'

April 12, 2016

Type Package

Title Morphometric analysis of digital elevation models.

Version 1.0

Date 2016-03-21

Author Sebastian Baumann

Maintainer Sebastian Baumann <sebastian.baumann@stud.sbg.ac.at>

Description R package for morphometric analysis of DEMs.

Depends rgdal, spatstat, maptools, rgeos, sp, raster, igraph

License GPL-2

R topics documented:

`lineaDEM-package` *Morphometric analysis of digital elevation models.*

Description

R package for morphometric analysis of DEMs.

Details

R package for morphometric analysis of objects of type RasterLayer. Major functions are a multi direction, multi scale roughness filter [toporough()] and a iterative hough transformation [hough.rough] to detect lineaments with a modified output of toporogh().

Author(s)

Sebastian Baumann

Maintainer: Sebastian Baumann <sebastian.baumann@stud.sbg.ac.at>

See Also

`raster`

Examples

```
data("work.data",package="lineaDEM") #import data
dem.mara<-work.data[[3]]
rough<-toporough(dem.mara,"max",11,"qu",0.5)
plot(rough[[1]]) #plot roughness map
```

---

ele.slope *Elevation vs. Slope*

Description

Calculates slope statistics for binned elevation data from DEMs.

Usage

```
ele.slope(r.dem, n.bin)
```

Arguments

`r.dem`	A digital elevation model of object type RasterLayer generated by the command raster() in the raster package.
`n.bin`	Numerical, >0, generates elevation intervals for which slope statistics are computed. The range of elevations is divided by n.bin to get the interval range.

Details

Calculates mean, median, 1st and 3rd quartile for slopes for elevation intervals. Slope is calculated by the command terrain() from the raster package.

Value

Matrix of computed slope statistics for elevation intervals.

col1	Low elevation of interval.
col2	Top elevation of interval.
col3	Mean slope.
col4	median slope
col5	1st quartile
col6	3rd quartile

Author(s)

Sebastian Baumann

See Also

raster

Examples

```
data("work.data",package="lineaDEM") #import data
dem.austria<-work.data[[1]]
slope.stat<-ele.slope(dem.austria,50)
plot((slope.stat[,1]+slope.stat[,2])/2,slope.stat[,3],
xlab="elevation [m]",ylab="slope [m/m]") #plot mean slope
lines((slope.stat[,1]+slope.stat[,2])/2,slope.stat[,3])

## The function is currently defined as
function (r.dem, n.bin)
{
    r.slope <- terrain(r.dem, opt = "slope")
    r.dem <- r.slope/r.slope * r.dem
    na.max <- function(x) ifelse(!all(is.na(x)), max(x, na.rm = T),
        NA)
    na.min <- function(x) ifelse(!all(is.na(x)), min(x, na.rm = T),
        NA)
    min.ele <- na.min(r.dem[])
    max.ele <- na.max(r.dem[])
    range.ele <- max.ele - min.ele
    ele.interval <- seq(min.ele, max.ele, range.ele/n.bin)
    ele <- na.omit(r.dem[])
    slo <- na.omit(r.slope)
    M.ele.slo <- matrix(NA, nrow = (length(ele.interval) - 1),
        ncol = 7)
    colnames(M.ele.slo) <- c("low", "high", "mean", "median",
        "25% percentile", "75% percentile", "n")
    for (i in c(0:(length(ele.interval) - 1))) {
        stat.summary <- summary(slo[ele >= ele.interval[i] &
            ele < ele.interval[i + 1]])
        M.ele.slo[i, 7] <- as.numeric(length(slo[ele >= ele.interval[i] &
```

```
            ele < ele.interval[i + 1]]))
        M.ele.slo[i, 6] <- as.numeric(stat.summary["3rd Qu."])
        M.ele.slo[i, 5] <- as.numeric(stat.summary["1st Qu."])
        M.ele.slo[i, 4] <- as.numeric(stat.summary["Median"])
        M.ele.slo[i, 3] <- as.numeric(stat.summary["Mean"])
        M.ele.slo[i, 2] <- as.numeric(ele.interval[i])
        M.ele.slo[i, 1] <- as.numeric(ele.interval[i + 1])
    }
    return(M.ele.slo)
}
```

`extrema` *Local minimum distance extrema of RasterLayers.*

Description

Simple function to compute local extrema (maximum or minimum) of RasterLayers with a minimum distance threshold. Uses command focal() of the raster package.

Usage

```
extrema(r, ngbr, type)
```

Arguments

`r`	Object of type RasterLayer generated by the command raster() in the raster package.
`ngbr`	Odd number defines the used moving window size (n*n) and therefore the minimum distance of two local extrema. The minimum distance is (ngbr²+1)/2 *resolution.
`type`	"min" or "max", for local minima or maxima.

Details

Local extrama are defined by the position of the extreme value within a neighborhood. If the position of the extreme value within a neighborhood is in the center of the neighborhood, than the cell represents a local extrema. The minimum distance between two extrema is defined by the moving window size: (ngbr²+1)/2 *resolution. Depending on your x and y resolution the minimum distance might be different in x and y direction.

Value

Object of type RasterLayer with cell value 1 for local extrema and NA for the rest.

Author(s)

Sebastian Baumann

Examples

```
#searching for the highest peaks in Austria.
data("work.data",package="lineaDEM") #import data
dem.austria<-work.data[[1]]
r.peaks<-extrema(dem.austria,13,"max") #calculate local maxima
p.peaks<-xyFromCell(r.peaks,which(!is.na(r.peaks[]))) #xy of peaks
#eliminate edge pixels
adj<-adjacent(dem.austria,which(!is.na(r.peaks[])),directions=8,sorted=TRUE,id=TRUE)
adj.values<-extract(dem.austria,adj[,3])
p.peaks<-p.peaks[-unique(adj[which(is.na(adj.values)),1]),]
plot(dem.austria) #plot dem
points(p.peaks,pch=20) #plot highest peaks

## The function is currently defined as
function (r, ngbr, type)
{
    mid.coor <- (ngbr^2 + 1)/2
    if (type == "min") {
        rp <- focal(r, matrix(1, ncol = ngbr, nrow = ngbr), which.min)
    }
    else {
        rp <- focal(r, matrix(1, ncol = ngbr, nrow = ngbr), which.max)
    }
    rp[] <- ifelse(rp[] == mid.coor, 1, NA)
    return(rp)
  }
```

`hough.rough` *Multiple hough transformation for automated lineament detection for thinned raster maps.*

Description

A straight line detection designed for the thinned output of toporough().

Usage

```
hough.rough(r.th, r.th.points, r.dire, r.rough, angle, bin, dist.th, angle.th)
```

Arguments

`r.th`	RasterLayer of thresholded multi roughness index map from toporough.
`r.th.points`	RasterLayer of thresholded multi roughness index map from toporough() thinned by extrema.
`r.dire`	RasterLayer of perpendicular orientations from toporough().
`r.rough`	RasterLayer of multi roughness index (mri) map from toporough().
`angle`	Numerical, 0 <= angle <= 90. Maximum abberation angle from r.dire. See details.

bin	Numerical, 0 <= angle <= 180. Defines angle interval, usually between 0.5 and 5. See details
dist.th	Numerical, >0, usually 3 to 10 times resolution of input RasterLayer. Determines by distance if a point of r.th.points builds a detected lineament.
angle.th	Numerical, 0 < angle <= 90. Determines by orientation if a point of r.th.points builds a detected lineament.

Details

Multiple hough transformation designed for a modified output of toporough(): The multi roughness index output of toporough() is thresholded and afterwards thinned by extrema(). The perpendicular orientation of the orientation map of toporough() is calculated. The thinned points within the domains defined by r.th are used to perform a iterative hough transformation. Depending on angle and bin for each of these points different orientated lines through these points are generated and transformed to the hough space. In the hough space these lines are represented as points. In the hough space the point density of these points representing lines is calculated with the command density.ppp() from the package spatstat. As weights for density.ppp() the multi roughness index is used. As standard deviation of isotropic gaussian smoothing kernel (sigma) density.ppp() 1/n(points) is used. The global maxima of the points density map represents a lineament. This lineament is retransformed to the xy space and depending on dist.th and angle.th points of r.th.points with a distance less then dist.th and within the angle threshold (angle.th) are identified as builders of the lineament. These points are excluded from r.th.points and the process is restarted without these points. If only one point is left in r.th.points or no lineaments are detected anymore the process is ended. The whole porcess is repeated for other domains of r.th. The lineaments are cut to the extent of r.th, the thresholded multi roughness index map of toporogh().

Value

List of lineaments and vector of roughness parameters.

comp1	List of points defining lineaments in 2x2 matrices. col1= X, col2= Y
comp2	Vector of roughness parameters. Mean of all mri values of all points assigned to a certain line

Author(s)

Sebastian Baumann

References

Hough, P.V.C. Method and means for recognizing complex patterns, U.S. Patent 3,069,654, Dec. 18, 1962

See Also

spatstat, raster

Examples

```
data("work.data",package="lineaDEM") #import data
dem.mara<-work.data[[3]]
rough<-toporough(dem.mara,"max",11,"sq",0.09)
rough.th<-rough[[1]]
rough.th[]<-ifelse(rough.th[]>0.03,rough.th[],NA) #apply threshold
rough.th.points<-extrema(rough.th,3,"max") #thin rough.th to point pattern
rough[[4]][]<-ifelse(rough[[4]][]>90,rough[[4]][]-90,
ifelse(rough[[4]][]<=90,rough[[4]][]+90,NA)) #calculate perpendicular oriantations
lineaments<-hough.rough(rough.th,rough.th.points,rough[[4]],rough[[1]],90,1,100,60)
plot(rough.th)
for (i in c(1:length(lineaments[[1]])))
{
plot(lineaments[[1]][[i]],add=TRUE)
}


## The function is currently defined as
function (r.th, r.th.points, r.dire, r.rough, angle, bin, dist.th,
    angle.th)
{
    r.th.for.clump <- r.th
    r.th.for.clump[] <- ifelse(is.na(r.th[]), 0, 1)
    r.th.clump <- clump(r.th.for.clump)
    r.points.clump <- r.th.points/r.th.points * r.th.clump
    data.comp <- matrix(NA, ncol = 6, nrow = sum(!is.na(r.th.points[])))
    colnames(data.comp) <- c("x", "y", "id_area", "dire", "rough",
        "id_line")
    points <- xyFromCell(r.points.clump, which(!is.na(r.points.clump[])))
    data.comp[, 1] <- points[, 1]
    data.comp[, 2] <- points[, 2]
    data.comp[, 3] <- extract(r.points.clump, points)
    data.comp[, 4] <- extract(r.dire, points)
    data.comp[, 5] <- extract(r.rough, points)
    id.area.counts <- as.data.frame(table(data.comp[, 3]))
    id.use <- as.numeric(as.vector(id.area.counts[id.area.counts[,
        2] > 1, 1]))
    res.max <- max(res(r.th))
    dif.angles <- seq(-angle + bin, angle, bin)
    diag = sqrt(r.th@extent@xmax^2 + r.th@extent@ymax^2) * 2
    linessd <- list()
    rough.pointssd <- list()
    rough.pointss <- list()
    liness <- list()
    for (k in id.use) {
        zone <- r.th.clump
        zone[] <- ifelse(zone[] == k, 1, 0)
        zone[] <- ifelse(is.na(zone[]), 0, zone[])
        if (sum(zone[] == 1) > 10) {
            ext.zone <- extend(zone, c(res.max, res.max), value = 0)
            vec.hull <- tryCatch(rasterToContour(ext.zone), error = function(e) e)
            if (inherits(vec.hull, "error"))
```

```
    next
points <- data.comp[data.comp[, 3] == k, 1:2]
dires <- data.comp[data.comp[, 3] == k, 4]
roughs <- data.comp[data.comp[, 3] == k, 5]
points[, 1] <- points[, 1] + ((-r.th@extent@xmin -
    r.th@extent@xmax)/2)
points[, 2] <- points[, 2] + ((-r.th@extent@ymin -
    r.th@extent@ymax)/2)
M.angles.lin <- matrix(NA, ncol = length(dif.angles),
    nrow = length(dires))
for (i in c(1:length(dif.angles))) {
    M.angles.lin[, i] <- dires + dif.angles[i]
}
M.angles.lin <- ifelse(M.angles.lin < 0, M.angles.lin +
    180, M.angles.lin)
M.angles.lin <- ifelse(M.angles.lin > 180, M.angles.lin -
    180, M.angles.lin)
x2.dires <- cos(M.angles.lin * pi/180) + points[,
    1]
y2.dires <- sin(M.angles.lin * pi/180) + points[,
    2]
M.angles.per <- M.angles.lin + 90
M.angles.per <- ifelse(M.angles.per >= 180, M.angles.per -
    180, M.angles.per)
nx.dires.per <- cos(M.angles.per * pi/180)
ny.dires.per <- sin(M.angles.per * pi/180)
x.inters <- (points[, 1] * y2.dires - points[, 2] *
    x2.dires) * (-nx.dires.per)/((points[, 1] - x2.dires) *
    (-ny.dires.per) - (points[, 2] - y2.dires) *
    (-nx.dires.per))
y.inters <- (points[, 1] * y2.dires - points[, 2] *
    x2.dires) * (-ny.dires.per)/((points[, 1] - x2.dires) *
    (-ny.dires.per) - (points[, 2] - y2.dires) *
    (-nx.dires.per))
phi <- sqrt(x.inters^2 + y.inters^2)
phi <- ifelse(y.inters < 0, -phi, phi)
data <- matrix(NA, ncol = 2, nrow = length(M.angles.per))
data[, 1] <- as.numeric(M.angles.per)
data[, 2] <- as.numeric(phi)
M.roughs <- matrix(NA, ncol = ncol(M.angles.per),
    nrow = nrow(M.angles.per))
for (i in c(1:ncol(M.roughs))) {
    M.roughs[, i] <- roughs
}
rough.pointss <- list()
liness <- list()
j = 1
while (length(points) > 2) {
    p <- SpatialPoints(matrix(c(as.numeric(M.angles.per),
      as.numeric(phi)), ncol = 2, nrow = length(phi)))
    p <- as.ppp(p)
    roughs <- as.numeric(M.roughs)
    d <- density.ppp(p, sigma = 1/nrow(points), weights = roughs,
```

```
  edge = T)
rd <- raster(d)
densi <- max(rd[])
max.dp <- which.max(rd[])
max.hough <- xyFromCell(rd, which.max(rd[]))
if (length(max.hough) > 2) {
  max.hough <- max.hough[1, ]
}
x.normal = (max.hough[2] * cos(max.hough[1] *
  pi/180))
y.normal = (max.hough[2] * sin(max.hough[1] *
  pi/180))
x2.normal <- x.normal + cos((max.hough[1] - 90) *
  pi/180) * diag
y2.normal <- y.normal + sin((max.hough[1] - 90) *
  pi/180) * diag
x3.normal <- x.normal - cos((max.hough[1] - 90) *
  pi/180) * diag
y3.normal <- y.normal - sin((max.hough[1] - 90) *
  pi/180) * diag
line <- spLines(matrix(c(x3.normal, x2.normal,
  y3.normal, y2.normal), ncol = 2, nrow = 2))
builder <- which((gDistance(SpatialPoints(points),
  line, byid = T) <= dist.th) & ((sqrt((dires -
  max.hough[1])^2)) > 90 - angle.th | (sqrt((dires -
  max.hough[1])^2)) < 90 + angle.th))
x2.normal <- x2.normal + (r.th@extent@xmin +
  r.th@extent@xmax)/2
x3.normal <- x3.normal + (r.th@extent@xmin +
  r.th@extent@xmax)/2
y2.normal <- y2.normal + (r.th@extent@ymin +
  r.th@extent@ymax)/2
y3.normal <- y3.normal + (r.th@extent@ymin +
  r.th@extent@ymax)/2
line <- spLines(matrix(c(x3.normal, x2.normal,
  y3.normal, y2.normal), ncol = 2, nrow = 2))
proj4string(line) <- projection(r.th)
inters.in <- gIntersection(vec.hull[1, ], line)
if (length(inters.in) > 1) {
  lines <- list()
  id.zw <- 1
  for (m in seq(1, length(inters.in@coords)/2,
    2)) {
    lines[[id.zw]] <- spLines((matrix(c(inters.in@coords[m,
      1], inters.in@coords[m + 1, 1], inters.in@coords[m,
      2], inters.in@coords[m + 1, 2]), ncol = 2,
      nrow = 2)))
    id.zw <- 1 + id.zw
  }
  points.dem <- points[builder, ]
  roughs.dem <- roughs[builder]
  if (length(points.dem) > 2) {
    points.dem[, 1] <- points.dem[, 1] + (r.th@extent@xmin +
```

```
                    r.th@extent@xmax)/2
                  points.dem[, 2] <- points.dem[, 2] + (r.th@extent@ymin +
                    r.th@extent@ymax)/2
                  o = 1
                  lines2 <- list()
                  n.points <- list()
                  dens <- list()
                  rough.points <- list()
                  for (n in c(1:length(lines))) {
                    if (sum((gDistance(SpatialPoints(points.dem),
                      lines[[n]], byid = T) <= dist.th)) >
                      1) {
                      lines2[[o]] <- lines[[n]]
                      n.points[[o]] <- sum((gDistance(SpatialPoints(points.dem),
                        lines[[n]], byid = T) <= dist.th))
                      dens[[o]] <- densi
                      builder2 <- which((gDistance(SpatialPoints(points.dem),
                        lines[[n]], byid = T) <= dist.th))
                      rough.points[[o]] <- mean(roughs.dem[builder2])
                      o = o + 1
                    }
                  }
                  liness[[j]] <- lines2
                  rough.pointss[[j]] <- rough.points
                }
              }
              dires <- dires[-builder]
              points <- points[-builder, ]
              M.angles.per <- M.angles.per[-builder, ]
              M.roughs <- M.roughs[-builder, ]
              phi <- phi[-builder, ]
              j = j + 1
            }
          }
          linessd[[k]] <- unlist(liness)
          rough.pointssd[[k]] <- unlist(rough.pointss)
        }
        lines.fine <- unlist(linessd)
        rough.points.final <- unlist(rough.pointssd)
        return(list(lines.fine, rough.points.final))
      }
```

`hypso` *Hypsometric analysis*

Description

Calculates hypsometric curves, relative hypsometric curves and hypsometric integral from DEMs.

Usage

```
hypso(r.dem, min.ele)
```

Arguments

r.dem	A digital elevation model of object type RasterLayer generated by the command raster() in the raster package.
min.ele	Numerical, minimum elevation - Elevations greater than min.ele are used to perform hypsometric analysis. If NA all values are used.

Value

comp1	Data frame, sorted from high to low elevations: col1: cumulative area, col2: absolute elevation values, col3: relative elevation values
comp2	Hypsometric integral

Author(s)

Sebastian Baumann

References

Strahler, Arthur N. "Hypsometric (area-altitude) analysis of erosional topography." Geological Society of America Bulletin 63.11 (1952): 1117-1142.

See Also

raster

Examples

```
data("work.data",package="lineaDEM") #import data
dem.austria<-work.data[[1]]
l.hypso<-hypso(dem.austria,NA) #use function
plot(l.hypso[[1]][,1],l.hypso[[1]][,2],
xlab="cumulative relative area",ylab="elevation [m]",type="n")
lines(l.hypso[[1]][,1],l.hypso[[1]][,2])


## The function is currently defined as
function (r.dem, min.ele)
{
    if (is.na(min.ele)) {
        elev <- r.dem[]
        elev <- na.omit(elev)
    }
    else {
        elev <- ifelse(r.dem[] >= min.ele, r.dem[], NA)
        elev <- na.omit(elev)
    }
```

```
  data <- matrix(NA, nrow = length(elev), ncol = 3)
  colnames(data) <- c("cumulative relative area", "elevation [m]",
      "relative elevation")
  data[, 2] <- sort(elev, decreasing = T)
  data[, 1] <- cumsum(as.numeric(c(1:length(elev))))/
  max(cumsum(as.numeric(c(1:length(elev)))))
  data[, 3] <- (data[, 2] - min(data[, 2]))/(max(data[, 2]) -
      min(data[, 2]))
  hi <- sum(data[, 3])/length(data[, 3])
  return(list(data, hi))
}
```

`toporough` *Multi directional and multi scale relief roughness filter.*

Description

Detects rough domains (mri) on digital elevation models and outputs additional paramatar as direction and widht of the structure.

Usage

```
toporough(dem, type, ngbr, nform, k.diff)
```

Arguments

`dem`	A digital elevation model of object type RasterLayer generated by the command raster() in the raster package.
`type`	"min" or "max". "min" corresponds to ridge structures, "max" to valley structures.
`ngbr`	Numerical, odd number - defines the used maximum moving window size (n*n). Smaller moving window sizes are calculated automatically.
`nform`	"circ" or "qu"- defines the shape of the moving window, circular or quadratic.
`k.diff`	Numerical - maximum allowed slope of a straight line connecting two edge points. If NA all slopes are allowed. See details.

Details

The multi roughness index (mri) is calculated as the vertical elevation difference (d) between the center cell and the different orientated straight lines connecting two edge cells of a neighborhood, divided by the horizontal distance (width) of the edge cells. Thus multiple roughness values depending on the neighborhood sizes and orientations of the edge connecting lines are generated for each cell and the maximum (valley) or minimum (ridge) values are extracted. The additional parameter k represents the maximum allowed slope of the horizontal distance line (width). Slope here is calculated by the horizontal distance and elevation difference.

Value

A list of raster maps.

comp1	Raster map of multi roughness index (mri).
comp2	Raster map of k values.
comp3	Raster map of d values.
comp4	Raster map of orientations.
comp5	Raster map of widths.

Warning

Different raster resolution in x and y direction: (xres+yres)/2 has been used as overall resolution.

Author(s)

Sebastian Baumann

See Also

raster

Examples

```
data("work.data",package="lineaDEM") #import data
dem.mara<-work.data[[3]]
rough<-toporough(dem.mara,"max",11,"qu",0.5)
plot(rough[[1]]) #plot roughness map

## The function is currently defined as
function (dem, type, ngbr, nform, k.diff)
{
    res <- (xres(dem) + yres(dem))/2
    if (yres(dem) != xres(dem)) {
        warning("Different raster resolution in x and y direction:
        (xres+yres)/2 has been used as overall resolution.")
    }
    n.cells.raster <- length(dem[])
    ngbr.sq <- ngbr^2
    n.neigh <- (ngbr - 1)/2
    mid.coor <- (ngbr + 1)/2
    M1.dist <- matrix(1:ngbr, ncol = ngbr, nrow = ngbr, byrow = T)
    M2.dist <- matrix(1:ngbr, ncol = ngbr, nrow = ngbr)
    M.dist <- sqrt((M2.dist - M1.dist[mid.coor, mid.coor]) *
        (M2.dist - M1.dist[mid.coor, mid.coor]) + (M1.dist -
        M2.dist[mid.coor, mid.coor]) * (M1.dist - M2.dist[mid.coor,
        mid.coor]))
    v.dist <- c(M.dist[1:((ngbr.sq - 1)/2)])
    M.point.x <- matrix(v.dist, ncol = ((ngbr.sq - 1)/2), nrow = n.cells.raster,
        byrow = T)
    M.point.x <- M.point.x * res
```

```
M.dem <- getValuesFocal(dem, ngb = ngbr, names = TRUE)
if (nform == "circ") {
    M.dist.circ <- M.dist
    M.dist.circ[M.dist.circ[] > (ngbr - 1)/2] <- NA
    v.dist.circ <- c(M.dist.circ[1:(ngbr.sq)])
    v.dist.circ[!is.na(v.dist.circ)] <- 1
    M.dem <- t(t(M.dem) * v.dist.circ)
}
na.max <- function(x) ifelse(!all(is.na(x)), max(x, na.rm = T),
    NA)
na.min <- function(x) ifelse(!all(is.na(x)), min(x, na.rm = T),
    NA)
M.point.k <- (M.dem[, (ngbr.sq):(((ngbr.sq + 1)/2) + 1)] -
    ((M.dem[, 1:(((ngbr.sq) - 1)/2)])))/(-M.point.x * 2)
if (!is.na(k.diff)) {
    M.point.k <- ifelse(M.point.k >= -k.diff & M.point.k <=
        k.diff, M.point.k, NA)
}
M.point.y <- M.dem[, (ngbr.sq):(((ngbr.sq + 1)/2) + 1)]
M.point.d <- M.point.y - ((M.point.k * (-M.point.x)))
M.point.d.ver <- (M.point.d - M.dem[, (ngbr.sq + 1)/2])/M.point.x/2
if (type == "min") {
    rough <- apply(M.point.d.ver, 1, na.min)
    rough.pos <- as.numeric(apply(M.point.d.ver, 1, which.min))
}
else {
    rough <- apply(M.point.d.ver, 1, na.max)
    rough.pos <- as.numeric(apply(M.point.d.ver, 1, which.max))
}
r.rough <- dem
r.rough[] <- rough
r.pos <- dem
r.pos[] <- rough.pos
r.k <- dem
r.k[] <- sqrt((M.point.k[cbind(1:nrow(M.point.k), rough.pos)])^2)
r.width <- dem
r.width[] <- M.point.x[cbind(1:nrow(M.point.x), rough.pos)] *
    2
r.d <- dem
r.d[] <- M.point.d[cbind(1:nrow(M.point.d), rough.pos)]
r.d <- r.d - dem
v.dire <- rep(c(rep(-1, (ngbr - 1)/2), rep(1, (ngbr + 1)/2)),
    ngbr)
M1.dire <- matrix(((ngbr - 1)/-2):((ngbr - 1)/2), ncol = ngbr,
    nrow = ngbr, byrow = T)
M2.dire <- (-1) * (matrix(((ngbr - 1)/-2):((ngbr - 1)/2),
    ncol = ngbr, nrow = ngbr))
M.dire <- ((M2.dire)/(sqrt(M1.dire^2 + M2.dire^2)))
M.dire <- acos(M.dire) * 180/pi
M.dire <- t(t(M.dire) * v.dire)
M.dire <- t(M.dire)
v.dire <- M.dire[1:((ngbr.sq - 1)/2)]
v.dire <- ifelse(v.dire < 0, v.dire + 180, v.dire)
```

```
  r.dire <- r.pos
  r.dire[] <- v.dire[r.pos[]]
  gc()
  return((c(r.rough, r.k, r.d, r.dire, r.width)))
}
```

`toporough.subset` *Multi directional and multi scale relief roughness filter for a subset of points.*

Description

Detects rough domains (mri) on a subset of points of a digital elevation model and outputs additional paramatar as direction and widht of the structure.

Usage

```
toporough.subset(dem, d.subset, type, ngbr, nform, k.diff)
```

Arguments

`dem`	See toporough.
`d.subset`	Matrix of X (col1) and Y(col2) coordinates of subset
`type`	See toporough.
`ngbr`	See toporough.
`nform`	See toporough.
`k.diff`	See toporough.

Details

See toporough.

Value

A matrix of extracted parameters and the subset coordinates. col1: X col2: Y col3: mri col4: k col5: d col6: orientation col7: width

`col1`	X coordinate.
`col2`	Y coordinate.
`col3`	Multi roughness index.
`col4`	k - parameter.
`col5`	d - parameter.
`col6`	Geographic orienatation.
`col6`	Width.

Author(s)

Sebastian Baumann

Examples

```
data("work.data",package="lineaDEM") #import data
dem.austria<-work.data[[1]]
accu.austria<-work.data[[2]]
track<-track.extrema(accu.austria,c(500000,220000),
9,"max","multi",NA) #generate subset
mri.sub<-toporough.subset(dem.austria,track,"max",51,"quad",NA)
plot(1:length(mri.sub[,1]),mri.sub[,6],
xlab="point number",ylab="orientation [°]")#plot n vs. orientation
lines(1:length(mri.sub[,1]),mri.sub[,6])

## The function is currently defined as
function (dem, d.subset, type, ngbr, nform, k.diff)
{
    res <- (xres(dem) + yres(dem))/2
    if (yres(dem) != xres(dem)) {
        warning("Different raster resolution in x and y direction:
        (xres+yres)/2 has been used as overall resolution.")
    }
    ngbr.sq <- ngbr^2
    n.neigh <- (ngbr - 1)/2
    mid.coor <- (ngbr + 1)/2
    m.ngb <- matrix(1, ngbr, ngbr)
    m.ngb[((ngbr + 1)/2), (ngbr + 1)/2] <- 0
    cell.sub <- cellFromXY(dem, d.subset)
    adj <- adjacent(dem, cell.sub, m.ngb, include = T, sorted = T)
    adj <- cbind(adj, vector(length = nrow(adj)))
    adj[, 3] <- extract(dem, adj[, 2])
    M.dem <- matrix(NA, nrow = length(cell.sub), ncol = ngbr.sq)
    for (i in c(1:length(cell.sub))) {
        M.dem[i, ] <- adj[adj[, 1] == cell.sub[i], 3]
    }
    n.cells.raster <- length(cell.sub)
    M1.dist <- matrix(1:ngbr, ncol = ngbr, nrow = ngbr, byrow = T)
    M2.dist <- matrix(1:ngbr, ncol = ngbr, nrow = ngbr)
    M.dist <- sqrt((M2.dist - M1.dist[mid.coor, mid.coor]) *
        (M2.dist - M1.dist[mid.coor, mid.coor]) + (M1.dist -
        M2.dist[mid.coor, mid.coor]) * (M1.dist - M2.dist[mid.coor,
        mid.coor]))
    v.dist <- c(M.dist[1:((ngbr.sq - 1)/2)])
    M.point.x <- matrix(v.dist, ncol = ((ngbr.sq - 1)/2), nrow = n.cells.raster,
        byrow = T)
    M.point.x <- M.point.x * res
    if (nform == "circ") {
        M.dist.circ <- M.dist
        M.dist.circ[M.dist.circ[] > (ngbr - 1)/2] <- NA
        v.dist.circ <- c(M.dist.circ[1:(ngbr.sq)])
        v.dist.circ[!is.na(v.dist.circ)] <- 1
```

```
        M.dem <- t(t(M.dem) * v.dist.circ)
    }
    na.max <- function(x) ifelse(!all(is.na(x)), max(x, na.rm = T),
        NA)
    na.min <- function(x) ifelse(!all(is.na(x)), min(x, na.rm = T),
        NA)
    M.point.k <- (M.dem[, (ngbr.sq):(((ngbr.sq + 1)/2) + 1)] -
        ((M.dem[, 1:(((ngbr.sq) - 1)/2)])))/(-M.point.x * 2)
    if (!is.na(k.diff)) {
        M.point.k <- ifelse(M.point.k >= -k.diff & M.point.k <=
            k.diff, M.point.k, NA)
    }
    M.point.y <- M.dem[, (ngbr.sq):(((ngbr.sq + 1)/2) + 1)]
    M.point.d <- M.point.y - ((M.point.k * (-M.point.x)))
    M.point.d.ver <- (M.point.d - M.dem[, (ngbr.sq + 1)/2])/M.point.x/2
    if (type == "min") {
        rough <- apply(M.point.d.ver, 1, na.min)
        rough.pos <- as.numeric(apply(M.point.d.ver, 1, which.min))
    }
    else {
        rough <- apply(M.point.d.ver, 1, na.max)
        rough.pos <- as.numeric(apply(M.point.d.ver, 1, which.max))
    }
    data <- matrix(NA, ncol = 7, nrow = n.cells.raster)
    colnames(data) <- c("X", "Y", "mri", "k", "d", "dire", "width")
    data[, 1] <- d.subset[, 1]
    data[, 2] <- d.subset[, 2]
    data[, 3] <- rough
    data[, 4] <- sqrt((M.point.k[cbind(1:nrow(M.point.k), rough.pos)])^2)
    data[, 5] <- M.point.d[cbind(1:nrow(M.point.d), rough.pos)] -
        extract(dem, d.subset)
    v.dire <- rep(c(rep(-1, (ngbr - 1)/2), rep(1, (ngbr + 1)/2)),
        ngbr)
    M1.dire <- matrix(((ngbr - 1)/-2):((ngbr - 1)/2), ncol = ngbr,
        nrow = ngbr, byrow = T)
    M2.dire <- (-1) * (matrix(((ngbr - 1)/-2):((ngbr - 1)/2),
        ncol = ngbr, nrow = ngbr))
    M.dire <- ((M2.dire)/(sqrt(M1.dire^2 + M2.dire^2)))
    M.dire <- acos(M.dire) * 180/pi
    M.dire <- t(t(M.dire) * v.dire)
    M.dire <- t(M.dire)
    v.dire <- M.dire[1:((ngbr.sq - 1)/2)]
    v.dire <- ifelse(v.dire < 0, v.dire + 180, v.dire)
    data[, 6] <- v.dire[rough.pos]
    data[, 7] <- M.point.x[cbind(1:nrow(M.point.x), rough.pos)] *
        2
    gc()
    return(data)
}
```

`track.extrema` *Tracks on RasterLayers.*

Description

Computes tracks on RasterLayers starting from a defined point. Uses the command adjacent() from the package raster.

Usage

```
track.extrema(r, startpoint, ngbr, type, method, endcon)
```

Arguments

r	Object of type RasterLayer generated by the command raster() in the raster package.
startpoint	A vector of X and Y coordinates of the starting point.
ngbr	Numerical, odd number - defines the n*n neighborhood to search for the next point on the track.
type	"min" or "max", for finding the minimum or maximum cell value within the moving window.
method	"multi" or "single": "multi": all cells of the prior moving window are ignored in the next steps. "single": only the used (maximum or minmum) cell is ignored in the next steps.
endcon	Numerical - endcondition to stop the track. If endcon is a number, track stops if the value of a computed track cell is lower (type="min") or greater (type="max") than endcon. If endcon is NA, track stops if the value of a computed track cell is greater (type="min") or lower (type="max") than any other trackvalue.

Details

track.max performs a iterate minimum or maximum search within a n*n neighborhood defined by a neighborhood size on RasterLayers. It ends if the endcon criteria is met or if a NA value occurs within the neighborhood.

Value

Returns a Matrix with X (col1) and Y (col2) coordinates of points building the track.

Author(s)

Sebastian Baumann

See Also

raster

Examples

```
data("work.data",package="lineaDEM") #import data
accu.austria<-work.data[[2]]
track<-track.extrema(accu.austria,c(500000,220000),9,"max","multi",NA)
plot(accu.austria)
lines(track,col="blue")

## The function is currently defined as
function (r, startpoint, ngbr, type, method, endcon)
{
    if (type == "min") {
        ext = which.min
    }
    if (type == "max") {
        ext = which.max
    }
    track <- vector()
    adj.all <- vector()
    adj <- vector()
    values <- vector()
    max.cell <- NA
    window <- matrix(1, ncol = ngbr, nrow = ngbr)
    mid.coor <- (ngbr^2 + 1)/2
    window[mid.coor] <- 0
    startcell <- cellFromXY(r, startpoint)
    i = 1
    while (!(any(is.na(values)))) {
        track[i] <- startcell
        adj <- adjacent(r, startcell, window)
        if (length(which(adj[, 2] %in% adj.all)) != 0) {
            adj <- adj[-which(adj[, 2] %in% adj.all), ]
        }
        if (length(adj) == 0)
            break
        if (length(adj) > 2) {
            values <- (extract(r, adj[, 2]))
            ext.cell <- ext(extract(r, adj[, 2]))
            last.start.cell <- startcell
            startcell <- as.numeric(adj[ext.cell, 2])
            ext.value <- extract(r, startcell)
            last.ext.value <- extract(r, last.start.cell)
            if (method == "single") {
                adj.all <- c(adj.all, startcell, last.start.cell)
            }
            if (method == "multi") {
                adj.all <- c(adj.all, adj[, 2], startcell, last.start.cell)
            }
        }
        else {
            values <- (extract(r, adj[2]))
            ext.cell <- ext(extract(r, adj[2]))
            last.start.cell <- startcell
```

```
            startcell <- as.numeric(adj[ext.cell])
            ext.value <- extract(r, startcell)
            last.ext.value <- extract(r, last.start.cell)
            if (method == "single") {
                adj.all <- c(adj.all, startcell, last.start.cell)
            }
            if (method == "multi") {
                adj.all <- c(adj.all, adj[2], startcell, last.start.cell)
            }
        }
        i = i + 1
        if (is.numeric(endcon)) {
            if (type == "min") {
                if (ext.value < endcon)
                  break
            }
            if (type == "max") {
                if (ext.value > endcon)
                  break
            }
        }
        if (!is.numeric(endcon)) {
            if (type == "min") {
                if (ext.value > last.ext.value)
                  break
            }
            if (type == "max") {
                if (ext.value < last.ext.value)
                  break
            }
        }
    }
    track.xy <- xyFromCell(r, track)
    return(track.xy)
}
```

`work.data` *Datasets used in examples.*

Description

Data of class RasterLayer used in examples.

Usage

```
data("work.data")
```

Format

List of 3 Formal class 'RasterLayer' [package "raster"].

Details

List of 3 Formal class 'RasterLayer' [package "raster"].

1st: Digital elevation model of Austria. RasterLayer from the command getData('alt',country="AUT") of the raster package, bilinear interpolated to "+proj=tmerc +lat_0=0 +lon_0=13.33333333333333 +k=1 +x_0=450000 +y_0=-5000000 +ellps=bessel +towgs84=577.326,90.129,463.919,5.13" and a resolution of 1000 meter.

2nd: Drainage accumulation map of Austria. Derived from the provided digital elevation model of austria with a D8 in grass gis with the command r.watershed.

3rd: Digital elevation map of an area in Souh Africa close to Marakele National park. RasterLayer, SRTM 1 Arc", bilinear interpolated to +proj=utm +zone=35 +datum=WGS84 +units=m +no_defs +ellps=WGS84 and a resolution of 30 m.

Source

http://earthexplorer.usgs.gov/

Examples

```
data("work.data",package="lineaDEM") #import data
plot(work.data[[1]])
```