Package ‘MC2toPath’

February 19, 2015

Type Package

Title Translates information from netcdf files with MC2 output into inter-PVT transitions.

Version 0.0.16

Date 2014-05-16

Author Dave Conklin and Emilie Henderson

Maintainer Dave Conklin <david.conklin@mac.com>

Imports RNetCDF

Description Post processes MC2 output, especially for use by Path or ST-Sim. MC2 (short for ``MC1 version 2'') is a dynamic global vegetation model (en.wikipedia.org/wiki/DGVM). Path (essa.com/tools/path) and ST-Sim (www.apexrms.com) are state-and-transition model (STM) engines. MC2 has a user website at sites.google.com/site/mc1dgvmusers. Since 2001, MC1 has been used to simulate changes in natural vegetation due to climate change at scales from regional to global. In 2012, MC1 was reimplemented in C++ to make it faster and to reduce storage requirements. This newer version is referred to as MC2, an abbreviation of ``MC1 version 2''. Beginning in 2011, output from MC1 and MC2 has been used to inform regional state-and-transition model simulations by the U.S. Forest Service and the Washington State Department of Natural Resources. Projects to date have involved study areas in central Oregon, the Olympic Peninsula, the Blue Mountains ecoregion, southwestern Oregon, and southeastern Oregon. In the first of this series of projects, the netCDF output files from MC2 were manually post-processed, mostly in Excel, to produce input .csv files for the STM engines. Beginning with the second project, R scripts were used to automate the post-processing work. These R scripts have been collected into the MC2toPath R-package.

Collate AggregateFireFracs.R AggregateVegFracs.R ReportMeanVegChanges.R SaveFireAreaFracs.R SaveMatrix.R SaveRatios.R SaveSequence.R SaveVegFracs.R VTnames.R VegTypeChanges.R SaveVegChangeProbabilityMultipliers.R SaveFireProbabilityMultipliers.R

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2014-05-18 08:11:42

R topics documented:

`MC2toPath-package` *Translate MC2 output netcdf files into inter-PVT transition rates.*

Description

Translate MC2 output netcdf files into inter-PVT transition rates.

Details

Package:	MC2toPath
Type:	Package
Version:	0.0.16
Date:	2014-05-16
License:	GPL-2

Post processes MC2 output, especially for use by Path or ST-Sim. A detailed example is given in the document, "MakingThePATHmegamodel.pdf" in inst/doc.

Author(s)

Author: Dave Conklin and Emilie Henderson Maintainer: Dave Conklin<david.conklin@mac.com>

AggregateFireFracs *Lump fire occurrence data for multiple PVTs*

Description

Aggregates fire occurrence data for 2 or more PVTs, writing the result to the console and returning it as an array.

Usage

```
AggregateFireFracs(vegChangesLocal, fireAreaFracsLocal, vt2pvtlutLocal, pvts2aggregate)
```

Arguments

`vegChangesLocal`	A list of five items returned by VegTypeChanges().
`fireAreaFracsLocal`	An array of fire fractions returned by SaveFireProbabilityMultipliers().
`vt2pvtlutLocal`	The veg type to PVT lookup table.
`pvts2aggregate`	A list of the PVTs to be lumped together.

Value

Returns an array indexed by year containing the aggregated fire fraction for each year.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (vegChangesLocal, fireAreaFracsLocal, vt2pvtlutLocal,
    pvts2aggregate)
{
    years = vegChangesLocal[[2]]
    nYrs = length(years)
    vts = vegChangesLocal[[3]]
    nVTs = length(vts)
    ndxsOfVTs2aggregate = c()
    nPVTs = length(pvts2aggregate)
    pvts2aggregateNdx = 1
    while (pvts2aggregateNdx <= nPVTs) {
        tgtPVT = pvts2aggregate[[pvts2aggregateNdx]]
        found = FALSE
        vtNdx = 0
        while (vtNdx < nVTs && !found) {
            vtNdx = vtNdx + 1
            vt = vt2pvtlutLocal$VT[vtNdx]
            pvt = levels(vt2pvtlutLocal$PVT)[vt2pvtlutLocal$PVT[vtNdx]]
            cat(c("tgtPVT, vtNdx, vt, pvt = ", tgtPVT, vtNdx,
                vt, pvt, "\n"))
            if (pvt == tgtPVT) {
                ndxsOfVTs2aggregate = c(ndxsOfVTs2aggregate,
                  vtNdx)
                pvts2aggregateNdx = pvts2aggregateNdx + 1
                found = TRUE
                cat(c("found vt ", vt, " at vtNdx ", vtNdx, " for tgtPVT ",
                  tgtPVT, "\n"))
            }
        }
        stopifnot(found)
    }
    stopifnot(length(ndxsOfVTs2aggregate) == nPVTs)
    aggVTfracs = array(0, nYrs)
    aggFireAreaFracs = array(0, nYrs)
    vtFracs = vegChangesLocal[[4]]
    stopifnot(dim(vtFracs)[1] == nVTs)
    stopifnot(dim(vtFracs)[2] == nYrs)
    for (aggNdx in 1:nPVTs) {
        vtNdx = ndxsOfVTs2aggregate[aggNdx]
        vt = vt2pvtlutLocal$VT[vtNdx]
        cat(c("vtNdx, vt = ", vtNdx, vt, "\n"))
        aggVTfracs = aggVTfracs + vtFracs[vtNdx, ]
        aggFireAreaFracs = aggFireAreaFracs + fireAreaFracsLocal[,
            vtNdx] * vtFracs[vtNdx, ]
    }
    aggFireFracs = array(0, nYrs)
    for (yrNdx in 1:nYrs) {
        if (aggVTfracs[yrNdx] > 0)
            aggFireFracs[yrNdx] = aggFireAreaFracs[yrNdx]/aggVTfracs[yrNdx]
        else aggFireFracs[yrNdx] = NA
    }
    return(aggFireFracs)
  }
```

AggregateVegFracs *Aggregate vegetation fractions*

Description

Sometimes it is useful to combine the vegetation fractions for several MC2 vegetation types, corresponding to different PVTs. AggregateVegFracs provides that capability.

Usage

```
AggregateVegFracs(vegChangesLocal, vt2pvtlutLocal, pvts2aggregate)
```

Arguments

`vegChangesLocal`

vegChangesLocal is a list of 5 named items: tgtfile, years, vts2keep, vtFracsReduced, changeFracsReduced tgtfile is the path and file name of the input netCDF file years is a vector of the calendar years represented in the change data, e.g. 2011, 2012, ... vts2keep is a vector of the VTYPEs occurring in the data. VTYPEs which never occur, such as tropical vegetation types in the Washington Coast Range, are omitted from vts2keep. vtFracsReduced is a 2-dimensional matrix containing for each vegetation type for each year the fraction of the total number of cells which has the given vegetation type in the given year changeFracsReduced is a 3-dimensional matrix containing the vegetation data.

`vt2pvtlutLocal` vt2pvtlutlocal ("VTYPE to PVT lookup table") is a data frame with 3 columns: VT, PVT, and Stratum. The VT column has the MC2 VTYPE integer value. The PVT column has a corresponding 3-letter potential vegetation type abbreviation such as "fdg", "fvg", etc. The Stratum column has a corresponding 7 character ILAP VDDT model name such as "WCR_fdg", "WCR_fvg".

`pvts2aggregate` pvts2aggregate is a vector of 3-letter PVT abbreviations, e.g. "fdw", "fvg"

Value

Returns a vector of fractions, one for each calendar year in the input data. Each fraction represents the fraction of all the active gridcells which is occupied by any of the vegetation types associated with the PVTs in the pvts2aggregate list.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (vegChangesLocal, vt2pvtlutLocal, pvts2aggregate)
{
    years = vegChangesLocal[[2]]
    nYrs = length(years)
    vts = vegChangesLocal[[3]]
    nVTs = length(vts)
    ndxsOfVTs2aggregate = c()
    nPVTs = length(pvts2aggregate)
    pvts2aggregateNdx = 1
    while (pvts2aggregateNdx <= nPVTs) {
        tgtPVT = pvts2aggregate[[pvts2aggregateNdx]]
        found = FALSE
        vtNdx = 0
        while (vtNdx < nVTs && !found) {
            vtNdx = vtNdx + 1
            vt = vt2pvtlutLocal$VT[vtNdx]
            pvt = levels(vt2pvtlutLocal$PVT)[vt2pvtlutLocal$PVT[vtNdx]]
            cat(c("tgtPVT, vtNdx, vt, pvt = ", tgtPVT, vtNdx,
        vt, pvt, "\n"))
      if (pvt == tgtPVT) {
        ndxsOfVTs2aggregate = c(ndxsOfVTs2aggregate,
         vtNdx)
        pvts2aggregateNdx = pvts2aggregateNdx + 1
        found = TRUE
        cat(c("found vt ", vt, " at vtNdx ", vtNdx, " for tgtPVT ",
         tgtPVT, "\n"))
      }
    }
    stopifnot(found)
  }
  stopifnot(length(ndxsOfVTs2aggregate) == nPVTs)
  aggFracs = array(0, nYrs)
  vtFracs = vegChangesLocal[[4]]
  stopifnot(dim(vtFracs)[1] == nVTs)
  stopifnot(dim(vtFracs)[2] == nYrs)
  for (aggNdx in 1:nPVTs) {
    vtNdx = ndxsOfVTs2aggregate[aggNdx]
    vt = vt2pvtlutLocal$VT[vtNdx]
    cat(c("vtNdx, vt = ", vtNdx, vt, "\n"))
    aggFracs = aggFracs + vtFracs[vtNdx, ]
  }
  return(aggFracs)
 }
```

`ReportMeanVegChanges` *Report Mean Vegetation Changes*

Description

Writes to the console, the mean values over all the years in the data, of the probabilities of transitions between vegetation types.

Usage

```
ReportMeanVegChanges(baseCalib, VTs, vtFracs, changeFracs, vtXpvt = data.frame(NULL))
```

Arguments

`baseCalib` The name of the base calibration, e.g. "GLOBAL", "CONUS".

`VTs` A vector of integer VTYPE values

`vtFracs` A matrix with one row for each element of VTs, and one column for each year of the simulation. The value of the matrix element is the fraction of all active gridcells occupied by the given vegetation type in the given year.

`changeFracs` A three-dimensional array. The third dimension varies over the years of the simulation. The first two dimensions vary over the length of VTs. The element value is the fraction of cells of the first veg type which transition to the second veg type in the given year.

`vtXpvt` vtXpvt is a dataframe, which defaults to NULL. When it is not null, it is used to crosswalk integer VTYPE values to 3-letter PVT abbreviations.

Value

Nothing meaningful is returned.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (baseCalib, VTs, vtFracs, changeFracs, vtXpvt = data.frame(NULL))
{
    vtNames = VTnames(baseCalib)
    nVT = length(VTs)
    nYrs = dim(changeFracs)[3]
    for (kSrc in 1:nVT) {
        meanFracOfAllCells = mean(vtFracs[kSrc, ])
        if (meanFracOfAllCells > 0) {
            if (length(vtXpvt) > 0) {
                cat(c("\n", "mean transition probabilities for transitions out of",
                  levels(vtXpvt$PVT)[vtXpvt$PVT[kSrc]], "...\n"))
                for (kDest in 1:nVT) if (kSrc != kDest) {
                  meanTransitionProbability = mean(changeFracs[kSrc,
                    kDest, ])
                  if (meanTransitionProbability > 0) {
                    cat(c(levels(vtXpvt$PVT)[vtXpvt$PVT[kSrc]],
                      "2", levels(vtXpvt$PVT)[vtXpvt$PVT[kDest]],
                      meanTransitionProbability, "\n"))
                  }
                }
            }
        }
    }
    cat("\n")
    cat(c("Mean values over ", dim(vtFracs)[2], " years:\n"))
    cat("frac of all cells, VTYPE\n")
    for (kSrc in 1:nVT) {
        meanFracOfAllCells = mean(vtFracs[kSrc, ])
        if (meanFracOfAllCells > 0) {
            vtName = vtNames[[VTs[kSrc]]]
            cat(c(meanFracOfAllCells, VTs[kSrc], vtName, "\n"))
        }
    }
  }
```

SaveFireAreaFracs *Save Fire Area Fractions*

Description

Write the fire area fractions out to a text file named "fireAreaFracs.csv".

Usage

```
SaveFireAreaFracs(vegChangesLocal, fireFracsLocal)
```

Arguments

vegChangesLocal	the list of 5 items returned by VegTypeChanges()
fireFracsLocal	the array of year-by-year fire fractions returned by SaveFireProbabilityMultipliers

Value

Returns a matrix fireAreaFracs[yrNdx, vtNdx] containing the same data which is written out to the text file.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (vegChangesLocal, fireFracsLocal)
{
    srcDataFile = vegChangesLocal[[1]]
    years = vegChangesLocal[[2]]
    nYr = length(years)
    VTs = vegChangesLocal[[3]]
    nVT = length(VTs)
    vegFracs = vegChangesLocal[[4]]
    outFile = "fireAreaFracs.csv"
    fireAreaFracs = matrix(0, nrow = nYr, ncol = nVT)
    cat(srcDataFile, file = outFile, append = FALSE)
    cat("\n", file = outFile, append = TRUE)
    cat("year", file = outFile, append = TRUE)
    for (vtNdx in 1:nVT) cat(c(", ", VTs[vtNdx]), file = outFile,
        append = TRUE)
    cat("\n", file = outFile, append = TRUE)
    for (yrNdx in 1:nYr) {
        cat(years[yrNdx], file = outFile, append = TRUE)
        for (vtNdx in 1:nVT) {
```

```
      fireAreaFracs[yrNdx, vtNdx] = vegFracs[vtNdx, yrNdx] *
        fireFracsLocal[yrNdx, vtNdx]
      cat(c(", ", fireAreaFracs[yrNdx, vtNdx]), file = outFile,
        append = TRUE)
    }
    cat("\n", file = outFile, append = TRUE)
  }
  return(fireAreaFracs)
}
```

`SaveFireProbabilityMultipliers`

Save fire probability multipliers as text files

Description

Creates two files: "fireFracs.csv" and "fireProbabilityMultipliers.txt". The .csv file is convenient for loading into Excel, and shows what fraction of each veg type is affected by fire in each year. The .txt file is formatted for loading into Path as a year-by-year probability multiplier file for transitions named WFNL, WFMS, and WFSR ("wildfire non-lethal", "wildfire moderate severity", and "wildfire stand-replacing").

Usage

```
SaveFireProbabilityMultipliers(infile, baseCalibration, vt2pvt_LUT)
```

Arguments

`infile` The path and filename of a "...year.nc" netCDF file containing the VTYPE and PART_BURN MC2 output variables.

`baseCalibration`
The name of the base calibration used by MC2, e.g. "CONUS" or "GLOBAL".

`vt2pvt_LUT` vt2pvt_LUT ("VTYPE to PVT lookup table") is a data frame with 3 columns: VT, PVT, and Stratum. The VT column has the MC2 VTYPE integer value. The PVT column has a corresponding 3-letter potential vegetation type abbreviation such as "fdg", "fvg", etc. The Stratum column has a corresponding 7 character ILAP VDDT model name such as "WCR_fdg", "WCR_fvg".

Value

Returns minFireFracs, a matrix. minFireFracs has one row for each year, and one column for each active veg type. The values in minFireFracs are the fraction of all the cells of a given veg type which had a fire in the given year.

Note

"PVT" is an acronym for "potential vegetation type". "ILAP" is an acronym for "Integrated Landscape Assessment Project", a research project carried out under the auspices of the US Forest Service in 2011-13. "VDDT" is an acronym for "Vegetation Dynamics Development Tool", a state-and-transition model engine, the predecessor of the Path state-and-transition model framework. "MC2" is a dynamic general vegetation model.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (infile, baseCalibration, vt2pvt_LUT)
{
    fP = open.nc(infile)
    VTYPE = var.get.nc(fP, "VTYPE")
    PART_BURN = var.get.nc(fP, "PART_BURN")
    YEAR = var.get.nc(fP, "year")
    nYrs = dim(VTYPE)[3]
    nVTall = length(VTnames(baseCalibration))
    vts2keepLUTndx = rep(0, times = nVTall)
    nStrata = length(vt2pvt_LUT$Stratum)
    for (i in 1:nStrata) {
        vts2keepLUTndx[vt2pvt_LUT$VT[i]] = i
    }
    vtCounts = matrix(nrow = nVTall, ncol = nYrs)
    vtFracs = matrix(nrow = nVTall, ncol = nYrs)
    fireFracs = matrix(nrow = nVTall, ncol = nYrs)
    fireFile = "fireFracs.csv"
    cat("\nFraction of cells in each veg type with simulated fires in each year\n",
        file = fireFile, append = FALSE)
    cat(infile, file = fireFile, append = TRUE)
    cat("\n\n", file = fireFile, append = TRUE)
    totFireFracThisYr = c(rep(0, nYrs))
    for (yr in 1:nYrs) {
        fireThisYr = c(rep(0, nVTall))
        vtCounts[, yr] = tabulate(VTYPE[, , yr], nVTall)
        nCellsActive = sum(vtCounts[, yr])
        if (yr > 1)
            stopifnot(nCellsActive == prev_nCellsActive)
        prev_nCellsActive = nCellsActive
        for (i in 1:dim(VTYPE)[1]) {
            for (j in 1:dim(VTYPE)[2]) {
                vt = VTYPE[i, j, yr]
                if (!is.na(vt) && PART_BURN[i, j, yr] > 0) {
                  stopifnot(1 <= vt && vt <= nVTall)
                  fireThisYr[vt] <- fireThisYr[vt] + 1
                }
            }
        }
        totFireFracThisYr[yr] = sum(fireThisYr)/nCellsActive
        print(c(YEAR[yr], totFireFracThisYr[yr]))
        for (vt in 1:nVTall) {
            if (vtCounts[vt, yr] > 0)
                fireFracs[vt, yr] = fireThisYr[vt]/vtCounts[vt,
                  yr]
            else fireFracs[vt, yr] = 0
        }
    }
    VTYPEf = factor(VTYPE, 1:nVTall, VTnames(baseCalibration))
    counts = tabulate(VTYPEf)
    VTofCol = c()
    nameOfCol = c()
    col = 0
    for (vt in 1:length(counts)) if (vts2keepLUTndx[vt] > 0) {
        VTofCol = c(VTofCol, vt)
        nameOfCol = c(nameOfCol, VTnames(baseCalibration)[vt])
        col = col + 1
    }
    nVTactive = length(VTofCol)
    stopifnot(nVTactive > 0)
    minFireFracs = matrix(nrow = nYrs, ncol = nVTactive)
    for (i in 1:nVTactive) minFireFracs[, i] = fireFracs[VTofCol[i],
        ]
    cat("cells, year", file = fireFile, append = TRUE)
    cat(", ", file = fireFile, append = TRUE)
    cat(nameOfCol, file = fireFile, sep = ", ", append = TRUE)
    cat(", ", file = fireFile, append = TRUE)
    cat("all", file = fireFile, append = TRUE)
    cat("\n", file = fireFile, append = TRUE)
    for (yr in 1:nYrs) {
        cat(nCellsActive, file = fireFile, sep = ", ", append = TRUE)
        cat(", ", file = fireFile, append = TRUE)
        cat(YEAR[yr], file = fireFile, sep = ", ", append = TRUE)
        cat(", ", file = fireFile, append = TRUE)
        cat(minFireFracs[yr, ], file = fireFile, sep = ", ",
            append = TRUE)
        cat(", ", file = fireFile, append = TRUE)
        cat(totFireFracThisYr[yr], file = fireFile, append = TRUE)
        cat("\n", file = fireFile, append = TRUE)
    }
    years = YEAR
    VTs = VTofCol
    nVT = length(VTs)
    nYrs = length(YEAR)
    cat(c(nVT, nVTactive, nYrs, length(VTofCol), dim(minFireFracs),
        VTofCol, "\n"))
    multiplierFile = "fireProbabilityMultipliers.txt"
    transitionTypes = c("WFNL", "WFMS", "WFSR")
    cat("\nMean fire probability over all years for each stratum\n")
    cat("VTYPE, stratum, mean fire probability per year\n")
    cat(infile, file = multiplierFile, append = FALSE)
    cat("\n", file = multiplierFile, append = TRUE)
```

```
    cat("kSrc, VTs[kSrc], vts2keepLUTndx[VTs[kSrc]],
            levels(vt2pvt_LUT$Stratum)[vt2pvt_LUT$Stratum[vts2keepLUTndx[VTs[kSrc]]]],
            meanFireProbability\n")
    for (kSrc in 1:nVTactive) {
        if (vts2keepLUTndx[VTs[kSrc]] < 1)
            next
      stratum = levels(vt2pvt_LUT$Stratum)[vt2pvt_LUT$Stratum[vts2keepLUTndx[VTs[kSrc]]]]
        meanFireProbability = mean(minFireFracs[, kSrc])
        cat(c(kSrc, VTs[kSrc], vts2keepLUTndx[VTs[kSrc]], stratum,
            meanFireProbability, "\n"))
        for (ttNdx in 1:length(transitionTypes)) {
            transitionType = transitionTypes[ttNdx]
            for (yr in 1:nYrs) {
                if (meanFireProbability == 0)
                  fireProbabilityMultiplier = 0
                else fireProbabilityMultiplier = minFireFracs[yr,
                  kSrc]/meanFireProbability
                outLine = paste(c(stratum, "\t\t", yr, "\tTemporal\t",
                  transitionType, "\t", fireProbabilityMultiplier),
                  collapse = "")
                cat(outLine, file = multiplierFile, append = TRUE)
                cat("\n", file = multiplierFile, append = TRUE)
            }
        }
    }
    cat(c(infile, baseCalibration, "SaveFireProbabilityMultipliers is finishing."))
    return(minFireFracs)
  }
```

SaveMatrix   *Save a matrix to a text file*

Description

Write out a matrix to a text file with row and column labels

Usage

```
SaveMatrix(startYear, localMatrix, outFile)
```

Arguments

startYear	the year to use as the first row label
localMatrix	the matrix to be written out
outFile	the file name and path of the file to be created

Value

Returns the incoming matrix with an extra column added on the left edge containing the years.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (startYear, localMatrix, outFile)
{
    nSeq = dim(localMatrix)[2]
    stopifnot(nSeq >= 1)
    matrixRows = dim(localMatrix)[1]
    stopifnot(matrixRows > 1)
    indexedSeq = matrix(0, nrow = nSeq, ncol = matrixRows + 1)
    appendFlag = FALSE
    for (ndx in 1:nSeq) {
        cat(c(startYear + ndx - 1), file = outFile, append = appendFlag)
        appendFlag = TRUE
        indexedSeq[ndx, 1] = startYear + ndx - 1
        for (row in 1:matrixRows) {
            if (is.na(localMatrix[row, ndx]))
                localMatrix[row, ndx] = 0
            cat(c(", ", localMatrix[row, ndx]), file = outFile,
                append = appendFlag)
            indexedSeq[ndx, row + 1] = localMatrix[row, ndx]
        }
        cat(c("\n"), file = outFile, append = appendFlag)
    }
    return(indexedSeq)
  }
```

SaveRatios *Normalize an array*

Description

Transforms an array with a non-zero mean into an array of corresponding values normalized to the mean, and writes them out to a text file.

Usage

```
SaveRatios(rawValues, outFile)
```

Arguments

`rawValues`	the original array of values - must have a non-zero mean
`outFile`	the file name and path of the output file

Value

the normalized array

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (rawValues, outFile)
{
    nRaw = length(rawValues)
    stopifnot(nRaw >= 1)
    meanVal = mean(rawValues, na.rm = TRUE)
    stopifnot(meanVal != 0)
    ratios = rawValues/meanVal
    appendFlag = FALSE
    for (ndx in 1:nRaw) {
        if (is.na(ratios[ndx]))
            ratios[ndx] = 0
        cat(c(ndx - 1, ", ", ratios[ndx], "\n"), file = outFile,
            append = appendFlag)
        appendFlag = TRUE
    }
    return(ratios)
  }
```

SaveSequence *Save a sequence of yearly values*

Description

Constructs a 2-column sequence of yearly values, and writes it to a text file.

Usage

```
SaveSequence(startYear, sequence, outFile)
```

Arguments

startYear	the calendar year of the first year in the sequence
sequence	an array of yearly values
outFile	the file name and path of the output file

Value

Returns a two-column matrix. The first column is the series of calendar years. The second column is the series of yearly values.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (startYear, sequence, outFile)
{
    nSeq = length(sequence)
    stopifnot(nSeq >= 1)
    indexedSeq = matrix(0, nrow = nSeq, ncol = 2)
    appendFlag = FALSE
    for (ndx in 1:nSeq) {
        if (is.na(sequence[ndx]))
            sequence[ndx] = 0
        cat(c(startYear + ndx - 1, ", ", sequence[ndx], "\n"),
            file = outFile, append = appendFlag)
        appendFlag = TRUE
        indexedSeq[ndx, 1] = startYear + ndx - 1
        indexedSeq[ndx, 2] = sequence[ndx]
    }
    return(indexedSeq)
  }
```

SaveVegChangeProbabilityMultipliers

Saves probability multipliers for Path to read in.

Description

Saves temporal inter-PVT multipliers in a format that can be read in to Path database

Usage

```
SaveVegChangeProbabilityMultipliers(vegChanges, project, climateChangeTransitionTypes,
    vt2pvtlut)
```

Arguments

`vegChanges` object created by the VegTypeChanges function

`project` String variable: Currently can be 'CONUS', or 'WCR'.

`climateChangeTransitionTypes`
Character vector: describing which PVT to PVT transition types to output.

`vt2pvtlut` vt2pvtlut ("VTYPE to PVT lookup table") is a data frame with 3 columns: VT, PVT, and Stratum. The VT column has the MC2 VTYPE integer value. The PVT column has a corresponding 3-letter potential vegetation type abbreviation such as "fdg", "fvg", etc. The Stratum column has a corresponding 7 character ILAP VDDT model name such as "WCR_fdg", "WCR_fvg".

Value

saves a file named 'vegChanges.csv' to the current working directory.

(no type defined yet. For CRAN, will probably want to assign an object type to this object) (Dave, can you explain what these items are?)

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (vegChanges, project, climateChangeTransitionTypes)
{
    srcDataFile = vegChanges[[1]]
    years = vegChanges[[2]]
    VTs = vegChanges[[3]]
    nVT = length(VTs)
    changeFracs = vegChanges[[5]]
    nYrs = dim(changeFracs)[3]
    stopifnot((nYrs + 1) == length(years))
    nTransitionTypes = length(climateChangeTransitionTypes)
    stopifnot(nTransitionTypes >= 1)
    multiplierFile = "vegChangeProbabilityMultipliers.txt"
    pvts <- vegChanges$vt2pvtlut[, 2]
    cat(srcDataFile, file = multiplierFile, append = FALSE)
    cat("\n", file = multiplierFile, append = TRUE)
    for (kSrc in 1:nVT) {
        for (kDest in 1:nVT) if (kSrc != kDest) {
            meanTransitionProbability = mean(changeFracs[kSrc,
                kDest, ])
            transitionType = paste(c(pvts[VTs[kSrc]], "2", pvts[VTs[kDest]]),
                collapse = "")
            iType = 0
            found = FALSE
            while (!found && iType < nTransitionTypes) {
                iType = iType + 1
                found = climateChangeTransitionTypes[iType] ==
                  transitionType
            }
            if (!found)
                next
            cat(c(transitionType, meanTransitionProbability,
                "\n"))
            for (yr in 1:nYrs) {
                if (meanTransitionProbability > 0)
                  transitionProbabilityMultiplier = changeFracs[kSrc,
                    kDest, yr]/meanTransitionProbability
                else transitionProbabilityMultiplier = 0
                cat(yr, file = multiplierFile, append = TRUE)
                cat("\tTemporal\t", file = multiplierFile, append = TRUE)
```

```
              cat(transitionType, file = multiplierFile, append = TRUE)
              cat("\t", file = multiplierFile, append = TRUE)
              cat(transitionProbabilityMultiplier, file = multiplierFile,
                append = TRUE)
              cat("\n", file = multiplierFile, append = TRUE)
           }
       }
    }
  }
```

SaveVegFracs — *Save the yearly series of veg type fractions*

Description

Save the yearly series of veg type fractions in a text file.

Usage

```
SaveVegFracs(vegChanges, outFile)
```

Arguments

`vegChanges`	A five-item list returned by VegTypeChanges()
`outFile`	the file name and path of the output file

Value

Returns TRUE.

Author(s)

Dave Conklin

Examples

```
## The function is currently defined as
function (vegChanges, outFile)
{
    startYear = vegChanges[[2]][1]
    SaveMatrix(startYear, vegChanges[[4]], outFile)
    return(TRUE)
  }
```

`vegChanges_step11` *Example data for step 11 in the MC2toPath package*

Description

The list object vegChanges_step11 is as produced by the call to VegTypeChanges() in step 11 of the example script in VegTypeChanges.Rd. The actual call to VegTypeChanges() in the example script has been commented out to save execution time in CRAN.

Usage

```
data(vegChanges_step11)
```

Format

a list object of length 6, containing: tgtFile, years, vts2keep, vtFracsReduced, changeFracsReduced, and vt2pvtlut

Source

Example step 11 run on WW2100_HadGEM2-ES85_year_sample.nc

References

doc/MakingThePATHmegamodel.pdf

`vegChanges_step3` *Example data for step 3 in the MC2toPath package*

Description

The list object vegChanges_step3 is as produced by the call to VegTypeChanges() in step 3 of the example script in VegTypeChanges.Rd. The actual call to VegTypeChanges() in the example script has been commented out to save execution time in CRAN.

Usage

```
data(vegChanges_step3)
```

Format

a list object of length 6, containing: tgtFile, years, vts2keep, vtFracsReduced, changeFracsReduced, and vt2pvtlut

Source

Example step 3 run on WW2100_HadGEM2-ES85_year_sample.nc

References

doc/MakingThePATHmegamodel.pdf

`VegTypeChanges` *Calculates transition rates.*

Description

Calculates transition rates from netcdf file output by MC2.

Usage

```
VegTypeChanges(tgtFile, baseCalibration, vtXpvt)
```

Arguments

`tgtFile` String: Full path to the netcdf output file.

`baseCalibration` String: currently ’CONUS’, or ’WCR’.

`vtXpvt` data frame, lookup table linking veg type indexes and PVT abbreviations.

Value

Returns a list object of length 6, containing:

tgtFile, years, vts2keep, vtFracsReduced, changeFracsReduced, and vt2pvtlut

Author(s)

Dave Conklin

Examples

```
## Please refer to the document "MakingThePATHmegamodel.pdf" in inst/doc.
## Step numbers given below refer to the step numbers in that document.

## The function is currently defined as
function (tgtFile, tgtVarName, baseCalibration, dontskipVTs,
    vt2pvtlut)
{
    tgtP = open.nc(tgtFile)
    tgtLonInfo = dim.inq.nc(tgtP, "lon")
    tgtLonDimID = tgtLonInfo$id
    tgtLatInfo = dim.inq.nc(tgtP, "lat")
    tgtLatDimID = tgtLatInfo$id
    tgtYrInfo = dim.inq.nc(tgtP, "year")
    tgtYrDimID = tgtYrInfo$id
    years = var.get.nc(tgtP, "year")
    tgtVarInfo = var.inq.nc(tgtP, tgtVarName)
```

```
tgtVarDimIds = tgtVarInfo$dimids
stopifnot(length(tgtVarDimIds) == 3)
stopifnot(tgtLonDimID == tgtVarDimIds[1])
stopifnot(tgtLatDimID == tgtVarDimIds[2])
stopifnot(tgtYrDimID == tgtVarDimIds[3])
tgtVar = var.get.nc(tgtP, tgtVarName)
tgtDim = dim(tgtVar)
stopifnot(length(tgtDim) == 3)
nCols = tgtDim[1]
nRows = tgtDim[2]
nYrs = tgtDim[3]
stopifnot(nYrs >= 2)
nCells = nCols * nRows
dim(tgtVar) = c(nCells, nYrs)
nVTall = length(VTnames(baseCalibration))
vtCounts = array(0, c(nVTall, nYrs))
for (yr in 1:nYrs) vtCounts[, yr] = tabulate(tgtVar[, yr],
    nbins = nVTall)
changePairs = array(FALSE, c(nVTall, nVTall))
for (yr in 2:nYrs) {
    for (cell in 1:nCells) {
        vtPrev = tgtVar[cell, yr - 1]
        stopifnot((1 <= vtPrev && vtPrev <= nVTall) || is.na(vtPrev))
        vtCurr = tgtVar[cell, yr]
        stopifnot((1 <= vtCurr && vtCurr <= nVTall) || is.na(vtCurr))
        if (!is.na(vtPrev) && !is.na(vtCurr))
            changePairs[vtPrev, vtCurr] = changePairs[vtPrev,
              vtCurr] || (vtPrev != vtCurr)
    }
}
vts2omit = rep(TRUE, times = nVTall)
for (vtPrev in 1:nVTall) {
    for (vtCurr in 1:nVTall) {
        if (changePairs[vtPrev, vtCurr])
            vts2omit[vtPrev] = FALSE
    }
}
rm(changePairs)
if (length(dontskipVTs) > 0) {
    for (k in 1:length(dontskipVTs)) vts2omit[dontskipVTs[k]] = FALSE
}
changeCounts = array(0, c(nVTall, nVTall, nYrs - 1))
changeFracs = array(0, c(nVTall, nVTall, nYrs - 1))
for (yr in 2:nYrs) {
    for (cell in 1:nCells) {
        vtPrev = tgtVar[cell, yr - 1]
        vtCurr = tgtVar[cell, yr]
        if (!is.na(vtPrev) && 1 <= vtPrev && vtPrev <= nVTall &&
            !is.na(vtCurr) && 1 <= vtCurr && vtCurr <= nVTall)
            changeCounts[vtPrev, vtCurr, yr - 1] = changeCounts[vtPrev,
              vtCurr, yr - 1] + 1
    }
    for (vtPrev in 1:nVTall) {
```

```
    for (vtCurr in 1:nVTall) {
      count = changeCounts[vtPrev, vtCurr, yr - 1]
      vtPrevTot = vtCounts[vtPrev, yr - 1]
      if (count > 0) {
       stopifnot(vtPrevTot >= 1)
       changeFracs[vtPrev, vtCurr, yr - 1] = count/vtPrevTot
      }
      else if (vtPrevTot == 0 && yr > 2) {
       changeFracs[vtPrev, vtCurr, yr - 1] = changeFracs[vtPrev,
        vtCurr, yr - 2]
      }
    }
  }
}
nVTreduced = nVTall - sum(vts2omit)
vtCountsReduced = array(0, c(nVTreduced, nYrs))
k = 0
for (vt in 1:nVTall) {
  if (!vts2omit[vt]) {
    k = k + 1
    vtCountsReduced[k, ] = vtCounts[vt, ]
  }
}
vtFracsReduced = array(0, c(nVTreduced, nYrs))
for (yr in 1:nYrs) {
  totCounts = sum(vtCountsReduced[, yr])
  vtFracsReduced = vtCountsReduced/totCounts
}
changeFracsReducedByRows = array(0, c(nVTall, nVTreduced,
  nYrs - 1))
vts2keep = rep(0, times = nVTreduced)
for (yr in 1:(nYrs - 1)) {
  k = 0
  for (vt in 1:nVTall) {
    if (!vts2omit[vt]) {
      k = k + 1
      vts2keep[k] = vt
      changeFracsReducedByRows[, k, yr] = changeFracs[,
       vt, yr]
    }
  }
}
changeFracsReduced = array(0, c(nVTreduced, nVTreduced, nYrs -
  1))
k = 0
for (vt in 1:nVTall) {
  if (!vts2omit[vt]) {
    k = k + 1
    changeFracsReduced[k, , ] = changeFracsReducedByRows[vt,
      , ]
  }
}
return(list(tgtFile = tgtFile, years = years, vts2keep = vts2keep,
        vtFracsReduced = vtFracsReduced, changeFracsReduced = changeFracsReduced,
        vt2pvtlut = vt2pvtlut))
  }

## Step 1
#ncdf.path = "MC2toPath/netcdf/WW2100_HadGEM2-ES85_year_sample.nc"
ncdf.path <- system.file("netcdf", "WW2100_HadGEM2-ES85_year_sample.nc", package = "MC2toPath")

## Step 2
base.calibration = "CONUS"

## Step 3
## This is what you would really do...
## vegChanges = VegTypeChanges(ncdf.th <- system.file("netcdf", "WW2100_HadGEM2-ES85_year_sample.nc", PACKAGE =
## but we do this instead to save execution time for CRAN...
data(vegChanges_step3)
vegChanges = vegChanges_step3

## Step 7
VTs = c(6, 7, 8, 10, 11, 12, 16, 22, 36)
PVTs = c("fmh", "fwi", "fdd", "fvg", "fdw", "fuc", "fto", "ftm", "fsi")
Strata = c("OWC_fmh", "OWC_fwi", "OSW_fdd", "OWC_fvg", "OWC_fdw", "OSW_fuc", "OWC_fto",
"OSW_ftm", "OWC_fsi")
vt2pvtlut = data.frame(VT=VTs, PVT=PVTs, Stratum=Strata)

## Step 8
climateChangeTransitionTypes = paste(rep(PVTs, length(PVTs)), "2", rep(PVTs,
each = length(PVTs)), sep="")

## Step 11
## This is what you would really do...
# vegChanges = VegTypeChanges(ncdf.path, base.calibration, vt2pvtlut)
## but we do this instead to save execution time for CRAN...
data(vegChanges_step11)
vegChanges = vegChanges_step11

## Step 12
SaveVegChangeProbabilityMultipliers(vegChanges, base.calibration,
climateChangeTransitionTypes, vt2pvtlut)

## Step 13
SaveFireProbabilityMultipliers(ncdf.path, base.calibration, vt2pvtlut)
```

VTnames *VTYPE names by base calibration*

Description

Given the name of the base calibration (e.g. CONUS, WCR), returns a list of the names of the potential vegetation types which MC2 can produce. The ith element of the list corresponds to VTYPE=i.

Usage

```
VTnames(baseCalibration)
```

Arguments

`baseCalibration`
: "CONUS" or "WCR". "GLOBAL" to be added eventually.

Value

Returns a list object consisting of text strings, one for each possible value of VTYPE (MC2's potential vegetation type output variable). For the CONUS base calibration, as of 1/23/14 VTYPE ranges from 1 thru 49, with several embedded unused values.

Author(s)

Dave Conklin

References

See MC2bigeography.xlsx in the MC2 Subversion repository.

Examples

```
## The function is currently defined as
function (baseCalibration)
{
    if (baseCalibration == "CONUS") {
        c("cold barren", "tundra aka alpine", "taiga-tundra",
            "boreal needleleaf forest", "boreal woodland", "subalpine forest",
            "maritime needleleaf forest", "temperate needleleaf forest",
            "temperate deciduous broadleaf forest", "cool mixed forest",
            "temperate warm mixed forest", "temperate needleleaf woodland",
            "temperate deciduous broadleaf woodland", "temperate cool mixed woodland",
            "temperate warm mixed woodland", "C3 shrubland",
            "C3 grassland", "temperate desert and semidesert",
            "subtropical needleleaf forest", "subtropical deciduous broadleaf forest",
            "warm evergreen broadleaf forest", "subtropical mixed forest",
           "subtropical needleleaf woodland", "subtropical deciduous broadleaf woodland",
            "subtropical evergreen broadleaf woodland", "subtropical mixed woodland",
            "C4 shrubland", "C4 grassland", "subtropical desert and semidesert",
            "tropical evergreen broadleaf forest", "tropical deciduous woodland",
            "tropical savanna", "unused33", "unused34", "tropical desert",
            "moist temperate needleleaf forest", "unused37",
            "subalpine meadow", "water and wetlands", "natural barren",
            "developed", "larch forest", "unused43", "unused44",
          "unused45", "unused46", "unused47", "unused48", "dry temperate needleleaf forest")
    }
    else if (baseCalibration == "WCR") {
        c("cold barren", "tundra aka alpine", "taiga-tundra",
            "boreal needleleaf forest", "boreal woodland", "subalpine forest",
            "maritime needleleaf forest", "temperate needleleaf forest",
            "temperate deciduous broadleaf forest", "cool mixed forest",
            "temperate warm mixed forest", "temperate needleleaf woodland",
            "temperate deciduous broadleaf woodland", "temperate cool mixed woodland",
            "temperate warm mixed woodland", "C3 shrubland",
            "C3 grassland", "temperate desert and semidesert",
            "subtropical needleleaf forest", "subtropical deciduous broadleaf forest",
            "warm evergreen broadleaf forest", "subtropical mixed forest",
           "subtropical needleleaf woodland", "subtropical deciduous broadleaf woodland",
            "subtropical evergreen broadleaf woodland", "subtropical mixed woodland",
            "C4 shrubland", "C4 grassland", "subtropical desert and semidesert",
            "tropical evergreen broadleaf forest", "tropical deciduous woodland",
            "tropical savanna", "unused33", "unused34", "tropical desert",
            "cool needleleaf forest", "unused37", "subalpine meadow",
            "water and wetlands", "natural barren", "developed",
            "larch forest", "Sitka spruce zone", "western hemlock zone",
            "Pacific silver fir zone", "mountain hemlock zone",
            "subalpine fir zone", "subalpine parkland zone")
    }
    else stopifnot(FALSE)
  }
```

Index