Package 'cepreader'

May 8, 2026

Title Read 'CEP' and Legacy 'CANOCO' Files

Version 1.3-0

Depends R (>= 3.6.0)

Imports Matrix

Description Read Condensed Cornell Ecology Program ('CEP') and legacy 'CANOCO' files into R data frames.

License MIT + file LICENCE

BugReports https://github.com/vegandevs/cepreader/issues

URL https://cran.r-project.org/,

https://github.com/vegandevs/cepreader/

NeedsCompilation yes

Author Jari Oksanen [aut, cre] (ORCID: <https://orcid.org/0000-0001-7102-9626>), Gavin L. Simpson [aut] (ORCID: <https://orcid.org/0000-0002-9084-8413>)

Maintainer Jari Oksanen <jhoksane@gmail.com>

Repository CRAN

Date/Publication 2026-03-04 14:10:03 UTC

Contents

readCEP *Reads a CEP (Canoco) data file*

Description

readCEP reads a file formatted by relaxed strict CEP format used by Canoco software, among others.

Usage

```
readCEP(file, maxdata = 10000, positive = TRUE, sparseMatrix = FALSE,
    long = FALSE, ...)
```

Arguments

file	File name (character variable).
maxdata	Maximum number of non-zero entries.
positive	Read only positive data entries (normal in community data).
sparseMatrix	Return data as a sparse matrix of **Matrix** package (see sparseMatrix). This option cannot be used together with long = TRUE.
long	Return data in "long" format as triplets of row name (row), column name (col) and data value (val). Only rows with positive value are included as default. This option cannot be used together with sparseMatrix = TRUE.
...	Other arguments passed to sparseMatrix.

Details

Cornell Ecology Programs (CEP) introduced several data formats designed for punched cards. One of these was the 'condensed strict' format which was adopted by popular software DECORANA and TWINSPAN. Later, Cajo ter Braak (1984) wrote Canoco based on DECORANA, where he adopted the format, but relaxed it somewhat (that's why I call it "relaxed strict" format). Further, he introduced a more ordinary "free" format, and allowed the use of classical Fortran style "open" format with fixed field widths. This function should be able to deal with all these Canoco formats, whereas it cannot read many of the traditional CEP alternatives.

All variants of CEP formats have:

- Two or three title cards, most importantly specifying the format (or word FREE) and the number of items per record (number of species and sites for "open" and "free" formats).
- Data in one of three accepted formats:
  1. Condensed format: First number on the line is the site identifier (an integer), and it is followed by pairs ("couplets") of numbers identifying the species and its abundance (an integer and a floating point number).
  2. Open Fortran format, where the first number on the line must be the site number, followed by abundance values in fields of fixed widths. Empty fields are interpreted as zeros.
  3. "Free" format, where the numbers are interpreted as abundance values. These numbers must be separated by blank space, and zeros must be written as zeros.

- Species and site names, given in Fortran format `(10A8)`: Ten names per line, eight columns for each.

With option `positive = TRUE` the function removes all lines and columns with zero or negative marginal sums in matrix like objects, or rows with non-positive value (`val`) in long format. In community data with only positive entries, this removes empty sites and species. If data entries can be negative, this ruins data, and such data sets should be read in with option `positive = FALSE`.

Value

Returns a data frame (default), where columns are species and rows are sites, or a similar data frame in long format where each entry is a triplet of row name, column name and data value (column names `row`, `col`, and `val`). Column and row names are taken from the CEP file, and changed into unique R names by `make.names` after stripping the blanks.

Alternatively the function can return a sparse matrix of **Matrix** package. This can give considerably saving in disk storage. However, typically the sparse matrix will be expanded to full dense matrix in community analysis. For instance, centred or standardized data matrices are dense. Moreover, some functions may be unable to analyse sparse matrices, but you must cast these to ordinary dense data matrices or data frames before the analysis.

Note

The function calls an external Fortran program to manipulate data so that they can be read into R. The function launches a separate program to read the data. This can trigger a warning with some security settings, and users may need to give permission for the operation.

If you transfer files between operating systems or platforms, you should always check that your file is formatted to your current platform. For instance, if you transfer files from Windows to Linux, you should change the files to `unix` format, or your session may crash when Fortran program tries to read the invisible characters that Windows uses as line terminators.

This function was included in **vegan** up to version 2.4-5. It was moved to a package of its own because compiled functions using Fortran I/O can interfere with compiled code written in other languages. This can disturb **vegan**, other loaded packages and packages that depend on **vegan**. Current versions of **vegan** have function `read.cep` that uses R code to read in "condensed" data. However, the R code is not as robust as the Fortran code in this package, and it can fail to read or read correctly several legacy files, and does not read "open" and "free" format data.

Author(s)

Jari Oksanen

References

Ter Braak, C.J.F. (1984–): CANOCO – a FORTRAN program for *cano*nical *c*ommunity *o*rdination by [partial] [detrended] [canonical] correspondence analysis, principal components analysis and redundancy analysis. *TNO Inst. of Applied Computer Sci., Stat. Dept. Wageningen, The Netherlands*.

Examples

```
## classic example
cepfile <- file.path(path.package("cepreader"), "testdata", "dune.spe")
## peek at the file structure
head(readLines(cepfile), n=10)
tail(readLines(cepfile), n=10)
## as a data frame
readCEP(cepfile)
## as a data frame in "long" format
head(readCEP(cepfile, long = TRUE))
## as a sparse matrix (Matrix package)
readCEP(cepfile, sparseMatrix = TRUE)
```

Index