

# Package ‘peptider’

October 14, 2022

**Title** Evaluation of Diversity in Nucleotide Libraries

**Version** 0.2.2

**Description** Evaluation of diversity in peptide libraries, including NNN, NNB, NNK/S, and 20/20 schemes. Custom encoding schemes can also be defined. Metrics for evaluation include expected coverage, relative efficiency, and the functional diversity of the library. Peptide-level inclusion probabilities are computable for both the native and custom encoding schemes.

**URL** <https://github.com/heike/peptider>

**BugReports** <https://github.com/heike/peptider/issues>

**Depends** R (>= 3.0.2)

**Imports** discreteRV (>= 1.2), plyr, dplyr

**Suggests** ggplot2

**License** GPL-3

**LazyData** true

**NeedsCompilation** no

**Author** Heike Hofmann [aut],  
Eric Hare [aut, cre],  
GGobi Foundation [aut]

**Maintainer** Eric Hare <erichare@iastate.edu>

**Repository** CRAN

**Date/Publication** 2015-09-16 10:05:27

## R topics documented:

BLOSUM80 . . . . .	2
codons . . . . .	3
coverage . . . . .	3
detect . . . . .	4
diversity . . . . .	5

efficiency . . . . .	5
encodingReduce . . . . .	6
generateCustom . . . . .	7
generateCustomLib . . . . .	7
generateCustomNei . . . . .	8
generateCustomProbs . . . . .	8
genNeighbors . . . . .	9
genNeighbors_reduced . . . . .	10
getChoices . . . . .	10
getCounts . . . . .	11
getNeighbors . . . . .	11
getNofNeighbors . . . . .	12
libBuild . . . . .	13
libscheme . . . . .	13
makowski . . . . .	14
ppeptide . . . . .	15
scheme . . . . .	16
schemes . . . . .	16
<b>Index</b>	<b>18</b>

---

BLOSUM80

*BLOSUM80 matrix*

---

## Description

The BLOSUM80 matrix, which stands for Blocks Substitution Matrix, defines log-odds scores for the ratio of the chance of two amino acids appearing in a sequence over the chance that the two amino acids appear in any sequence. Larger scores indicate a higher probability of substitutions. This matrix is used in order to compute sequences which are in the neighborhood of other sequences.

## Usage

`data(BLOSUM80)`

## Details

BLOSUM80 matrix

---

codons	<i>Compute the number of codon representations for a (vector of) peptide sequence(s)</i>
--------	--

---

**Description**

use this function for only a few peptide sequences. Any larger number of peptide sequences should be dealt with in the framework of the library scheme and the detect function.

**Usage**

```
codons(x, libscheme, flag = FALSE)
```

**Arguments**

x	(vector) of character strings of peptide sequences.
libscheme	library scheme under which neighbors are being calculated. this is only of importance, if method="dna"
flag	internal use only: Set to true if calling this from another function

**Value**

vector of numbers of codons

**Examples**

```
codons("APE", libscheme="NNK")  
codons("HENNING", libscheme="NNK")
```

---

coverage	<i>Coverage as expected number of peptides given all possible peptides</i>
----------	--

---

**Description**

Coverage of library of size N given random sampling from the pool of all possible peptides according to probabilities determined according to the library scheme.

**Usage**

```
coverage(k, libscheme, N, lib = NULL, variance = FALSE)
```

**Arguments**

k	length of peptide sequences
libscheme	Name (character vector) or definition (data frame) of scheme
N	size of the library
lib	library scheme
variance	return the variance instead of the expected value

**Value**

coverage index between 0 and 1

**Examples**

```
coverage(2, "NNN", 10^3)
coverage(2, "NNK", 10^3)
coverage(2, "2020", 10^3) ## 20/20 coverage is not 1 because of random sampling.
```

---

detect

*Detection probability in a single library of size N*

---

**Description**

The probability that at least one of a number of specific peptide sequences (e. g. the ‘best’ and closely related sequences) is contained in a library

**Usage**

```
detect(lib = libscheme("NNK", 7), size = 10^8)
```

**Arguments**

lib	library used in experiment, defaults to NNK with peptide length 7
size	size of the library, defaults to 10 <sup>8</sup>

**Value**

vector of detection probabilities for peptide sequences in each class

**Examples**

```
summary(detect())

require(ggplot2)
lib <- libscheme("NNK", 7)
qplot(detect(lib, size=10^8), weight=di, geom="histogram", data=lib$data)
```

---

diversity	<i>Diversity according to peptides paper (Sieber)</i>
-----------	---

---

**Description**

Diversity according to peptides paper (Sieber)

**Usage**

```
diversity(k, libscheme, N, lib = NULL, variance = FALSE)
```

**Arguments**

k	length of peptide sequences
libscheme	Name (character vector) or definition (data frame) of scheme
N	size of the library
lib	library scheme
variance	return the variance instead of the expected value

**Value**

Expected Diversity of the library

**Examples**

```
diversity(2, "NNN", 10^3)  
diversity(2, "NNK", 10^3)
```

---

efficiency	<i>Relative efficiency of a library</i>
------------	---

---

**Description**

Relative efficiency of a peptide library, defined as the ratio of expected diversity of a peptide library relative to its overall number of oligonucleotides

**Usage**

```
efficiency(k, libscheme, N, lib = NULL, variance = FALSE)
```

**Arguments**

k	length of peptide sequences
libscheme	Name (character vector) or definition (data frame) of scheme
N	size of the library
lib	library, if null, libscheme will be used to create it
variance	return the variance instead of the expected value

**Value**

relative efficiency index between 0 and 1

**Examples**

```
efficiency(3, "NNN", 10^2)
efficiency(3, "NNK", 10^2)
efficiency(3, "2020", 10^2) ## 20/20 efficiency is not 1 because of random sampling.
```

---

encodingReduce	<i>Reduce the regular encoding to an easier/faster format</i>
----------------	---

---

**Description**

Reduce the regular encoding to an easier/faster format

**Usage**

```
encodingReduce(class, libscheme)
```

**Arguments**

class	The peptide class
libscheme	The scheme to use

**Value**

Vector of reduced peptide encodings

---

generateCustom	<i>Generate peptide and library information for a given scheme</i>
----------------	--

---

**Description**

This function will generate library properties for a custom scheme. It is primarily intended to be used on <http://www.pelica.org>.

**Usage**

```
generateCustom(scheme_name = "custom", scheme_def = read.csv(file.choose()),  
              k = 1:20, n = 1:25, savefile = TRUE)
```

**Arguments**

scheme_name	The name of the resulting encoding scheme
scheme_def	A data frame containing encoding information for the scheme
k	peptide lengths to include
n	exponents of the library size to include
savefile	if true, save the results to an RData file

**Value**

TRUE upon completion of the script and output of the CSV files

**Examples**

```
## Not run:  
generateCustom()  
generateCustom(scheme_name = "NNN", scheme_def = scheme("NNN"))  
  
## End(Not run)
```

---

generateCustomLib	<i>For a given scheme, generate a dataset with the library information</i>
-------------------	--

---

**Description**

For a given scheme, generate a dataset with the library information

**Usage**

```
generateCustomLib(scheme_def, k = 1:20, n = 1:25)
```

**Arguments**

scheme_def	definition of the custom scheme
k	peptide lengths to include
n	exponents of the library size to include

**Value**

A data frame of library information

---

generateCustomNei	<i>For a given scheme, generate a dataset with the neighborhood information</i>
-------------------	---

---

**Description**

For a given scheme, generate a dataset with the neighborhood information

**Usage**

```
generateCustomNei(scheme_def, k = 1:20, n = 1:25)
```

**Arguments**

scheme_def	definition of the custom scheme
k	peptide lengths to include
n	exponents of the library size to include

**Value**

A data frame of neighborhood information

---

generateCustomProbs	<i>For a given scheme, generate a dataset with the peptide probabilities</i>
---------------------	--

---

**Description**

For a given scheme, generate a dataset with the peptide probabilities

**Usage**

```
generateCustomProbs(scheme_def, k = 1:20)
```



**Arguments**

scheme_def	definition of the custom scheme
k	peptide lengths to include

**Value**

A data frame of peptide probabilities

---

genNeighbors	<i>Calculate neighborhood distribution</i>
--------------	--

---

**Description**

Calculate distribution of neighbors under library scheme lib for peptide sequences of length k.

**Usage**

```
genNeighbors(sch, k)
```

**Arguments**

sch	library scheme
k	length of the peptide sequences

**Value**

dataset of peptide sequences: AA are amino acid sequences, c0 are codons for self representation, cr is the ratio of #neighbors in first degree neighborhood (not counting self representations) and #codons in self representation N1 is the number of neighbors in codon representation (including self representation)

**Examples**

```
genNeighbors(scheme("NNK"), 2)  
genNeighbors(scheme("2020"), 2)
```

---

genNeighbors\_reduced    *Calculate neighborhood distribution*

---

### Description

Calculate distribution of neighbors under library scheme lib for peptide sequences of length k.

### Usage

```
genNeighbors_reduced(sch, k)
```

### Arguments

sch	library scheme
k	length of the peptide sequences

### Value

dataset of peptide sequences: L are amino acid sequences, c0 are codons for self representation, cr is the ratio of #neighbors in first degree neighborhood (not counting self representations) and #codons in self representation N1 is the number of neighbors in codon representation (including self representation) s is the number of peptide sequences described by the label o is the number of peptide sequences reached by permutations

### Examples

```
genNeighbors_reduced(scheme("NNK"), 2)
genNeighbors_reduced(scheme("2020"), 2)
```

---

getChoices                    *Get the number of peptides that reduce to a particular reduced encoding*

---

### Description

Get the number of peptides that reduce to a particular reduced encoding

### Usage

```
getChoices(str)
```

### Arguments

str	The reduced encoding string
-----	-----------------------------

### Value

An integer of the possible number of peptides reducing to this encoding

---

getCounts	<i>Get the counts possible for each scheme and k</i>
-----------	--

---

**Description**

Get the counts possible for each scheme and k

**Usage**

```
getCounts(libscheme, k)
```

**Arguments**

libscheme	The scheme to use
k	Peptide length

**Value**

Character vector of possible counts for each class

---

getNeighbors	<i>Find all neighbors of degree one for a set of peptide sequences</i>
--------------	--

---

**Description**

first degree neighbors - a neighbor of a peptide is defined as a peptide sequence that differs in at most one amino acid from a given sequence. Additionally, we can restrict neighbors to regard only those sequences that have a certain minimal BLOSUM loading.

**Usage**

```
getNeighbors(x, blosum = 1)
```

**Arguments**

x	(vector) of character strings of peptide sequences.
blosum	minimal BLOSUM loading, defaults to 1 for positive loadings only

**Value**

list of neighbor sequences

**Examples**

```

getNeighbors("APE")
getNeighbors(c("HI", "APE"))
getNeighbors(c("HI", "EARNEST", "APE"), blosum=3)
## degree 2 neighbors:
unique(unlist(getNeighbors(getNeighbors("APE"))))

```

---

getNofNeighbors	<i>Compute the number of neighbor of degree one for a set of peptide sequences</i>
-----------------	--

---

**Description**

first degree neighbors - a neighbor of a peptide is defined as a peptide sequence that differs in at most one amino acid from a given sequence. Additionally, we can restrict neighbors to regard only those sequences that have a certain minimal BLOSUM loading. Use this function for only a few peptide sequences. Any larger number of peptide sequences will take too much main memory.

**Usage**

```
getNofNeighbors(x, blosum = 1, method = "peptide", libscheme = NULL)
```

**Arguments**

x	(vector) of character strings of peptide sequences.
blosum	minimal BLOSUM loading, defaults to 1 for positive loadings only
method	character string, one of "peptide" or "codon". This specifies the level at which the neighbors are calculated.
libscheme	library scheme under which neighbors are being calculated. this is only of importance, if method="dna"

**Value**

vector of numbers of neighbors

**Examples**

```

getNofNeighbors("APE")
getNofNeighbors(c("NEAREST", "EARNEST"))
getNofNeighbors("N")
getNofNeighbors("N", method="codon", libscheme="NNK")

```

---

libBuild	<i>Build peptide library of k-length sequences according to specified scheme</i>
----------	--

---

**Description**

Build peptide library of k-length sequences according to specified scheme

**Usage**

```
libBuild(k, libscheme, scale1 = 1, scale2 = 1)
```

**Arguments**

k	length of peptide sequences
libscheme	library scheme specifying classes of amino acids according to number of encodings last class is reserved for stop tags and other amino acids we are not interested in.
scale1	Scaling factor for first probs
scale2	Scaling factor for second probs

**Value**

library and library scheme used

**Examples**

```
user_scheme <- data.frame(class=c("A", "B", "C", "Z"),
                          aacid=c("SLR", "AGPTV", "CDEFGHIKMNQWY", "*"),
                          c=c(3,2,1,1))
user_library <- libBuild(3, user_scheme)
```

---

libscheme	<i>Get the specified library scheme</i>
-----------	---

---

**Description**

Get the specified library scheme

**Usage**

```
libscheme(schm, k = 1)
```

**Arguments**

schm	either a character vector giving the name of a built-in scheme, or a data frame consisting of the scheme definition
k	length of peptide sequences

**Value**

list consisting of a data frame of peptide classes, size of class, and its probabilities, and a list of additional information relating to the library scheme

**Examples**

```
libscheme("NNN")
libscheme("NNK", 2)

# Build a custom 20/20 library
custom <- data.frame(class = c("A", "Z"), aacid = c("SLRAGPTVIDEFHKNQYMW", "*"), c = c(1, 0))
libscheme(custom)
```

---

makowski

*Diversity index according to Makowski*


---

**Description**

The Diversity of a peptide library of length k according to Makowski and colleagues

**Usage**

```
makowski(k, libscheme)
```

**Arguments**

k	length of peptide sequences
libscheme	Name (character vector) or definition (data frame) of scheme

**Details**

Makowski and colleagues [Makowski, Soares 2003] present another approach by defining functional diversity. They provide the mathematical background to determine the quality of a peptide library based on the probability of individual peptides to appear. In an ideal case, where every peptide has the same frequency the functional diversity is at a maximum of 1. With increasingly skew distributions, this value drops towards a minimum of 0. It is mostly independent of the actual number of sequences in a library but reflects effects caused by the degeneration of the genetic code. In the genetic code the number of codons per amino acid varies from one to six. Therefore random DNA sequences are biased towards encoding peptides enriched in amino acids encoded more frequently, which results in skew distributions of peptide frequencies.

**Value**

diversity index between 0 and 1

**Examples**

```
makowski(2, "NNN")
makowski(3, "NNK")
makowski(3, "2020")
```

---

ppeptide

*Probability of detection of a peptide sequence*

---

**Description**

use this function for only a few peptide sequences. Any larger number of peptide sequences should be dealt with in the framework of the library scheme and the detect function.

**Usage**

```
ppeptide(x, libscheme, N)
```

**Arguments**

x	(vector) of character strings of peptide sequences.
libscheme	library scheme under which neighbors are being calculated.
N	number of valid DNA clones investigated

**Value**

probability of detection

**Examples**

```
ppeptide("APE", libscheme="NNK", N=10^8)
ppeptide("HENNING", libscheme="NNK", N=10^8)
```

---

scheme	<i>Get the specified library scheme definition</i>
--------	--

---

**Description**

Get the specified library scheme definition

**Usage**

```
scheme(name, file = NULL)
```

**Arguments**

name	name of the scheme as a character vector
file	CSV file hosting scheme definition, if provided

**Value**

a data frame of peptide classes, amino acids, and size of the classes corresponding to the selected scheme

**Examples**

```
scheme("NNN")  
scheme("NNK")
```

---

schemes	<i>Built-in library schemes for peptider</i>
---------	--

---

**Description**

This data set contains descriptions of amino acid classes several commonly used library schemes: NNN, NNB, NNK, 20/20, and variations of each in which Cysteine is not considered a viable amino acid.

**Usage**

```
data(schemes)
```



**Details**

## Built-in library schemes

The schemes are defined as:

NNN: All four bases (N = G/A/T/C) possible at all three positions in the codon. NNB: All four bases in the first two codon positions possible, the third position is restricted to G, T or C (=NB). NNK/S: All four bases in the first two codon positions possible, the third position is restricted to G/T (=NK) or two C/G (=S). 20/20: 20/20 describes the concept that DNA is assembled from prefabricated trimeric building blocks. This allows the generation of libraries from a predefined set of codons and thereby complete exclusion of Stop codons and other unwanted codons. NNN (-C): NNN with Cysteine ignored. NNB (-C): NNB with Cysteine ignored. NNK/SC (-C): NNK/S with Cysteine ignored. 20/20 (-C): 20/20 with Cysteine ignored.

The schemes differ in the number of used codons, ranging from 64 (NNN), 48 (NNB), 32 (NNK/S) to 20 or less (20/20). Coding schemes that allow varying ratios of codons/amino acid, result in libraries biased towards amino acids which are encoded more often. Further, the number of Stop codons that can lead to premature termination of the peptide sequence influences the performance of the library.

# Index

BLOSUM80, [2](#)

codons, [3](#)

coverage, [3](#)

detect, [4](#)

diversity, [5](#)

efficiency, [5](#)

encodingReduce, [6](#)

generateCustom, [7](#)

generateCustomLib, [7](#)

generateCustomNei, [8](#)

generateCustomProbs, [8](#)

genNeighbors, [9](#)

genNeighbors\_reduced, [10](#)

getChoices, [10](#)

getCounts, [11](#)

getNeighbors, [11](#)

getNofNeighbors, [12](#)

libBuild, [13](#)

libscheme, [13](#)

makowski, [14](#)

ppeptide, [15](#)

scheme, [16](#)

schemes, [16](#)