Package 'pgenlibr'

November 8, 2026

Type Package

Title 'PLINK' 2 Binary (.pgen) Reader

Version 0.7.1

Date 2026-11-08

Description A thin wrapper over 'PLINK' 2's core libraries which provides an R interface for reading .pgen files. A minimal .pvar loader and a basic .pgen writer are also included. Chang et al. (2015) <doi:10.1186/s13742-015-0047-8>.

Encoding UTF-8

License LGPL (>= 3)

Copyright This package includes sources of the libdeflate library owned by Eric Biggers, sources of the SIMDe library owned by Evan Nemerson, sources of the Zstd library owned by Meta Platforms, Inc., and sources of the pgenlib library owned by Christopher Chang.

BugReports https://github.com/chrchang/plink-ng/issues

Imports Rcpp (>= 1.0.1)

LinkingTo Rcpp

Suggests withr (>= 3.0.3), testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Christopher Chang [aut, cre],
Eric Biggers [ctb, cph] (Author of included libdeflate library),
Yann Collet [ctb] (Author of included Zstd library),
Meta Platforms, Inc. [cph] (Zstd library),
Evan Nemerson [ctb, cph] (Author of included SIMDe library),
Przemyslaw Skibinski [ctb] (Author of included Zstd library),
Nick Terrell [ctb] (Author of included Zstd library)

Maintainer Christopher Chang <chrchang@alumni.caltech.edu>

Repository CRAN

Date/Publication 2026-11-08 15:20:02 UTC

Contents

pgenlibr-package . . . 3
AlleleCodeBuf . . . 4
AppendAlleles . . . 5
AppendBiallelic . . . 6
AppendDosages . . . 6
BoolBuf . . . 7
Buf . . . 7
ClosePgen . . . 8
ClosePgenWriter . . . 8
ClosePvar . . . 9
GetAlleleCode . . . 9
GetAlleleCt . . . 10
GetMaxAlleleCt . . . 10
GetRawSampleCt . . . 11
GetVariantChrom . . . 11
GetVariantCt . . . 12
GetVariantId . . . 12
GetVariantPos . . . 13
GetVariantsById . . . 13
GetWrittenVariantCt . . . 14
HardcallPhasePresent . . . 14
HasSparse . . . 15
HasSparseHardcalls . . . 15
IntAlleleCodeBuf . . . 16
IntBuf . . . 16
NewPgen . . . 17
NewPgenWriter . . . 17
NewPvar . . . 18
Read . . . 19
ReadAlleles . . . 20
ReadHardcalls . . . 20
ReadIntList . . . 21
ReadList . . . 22
ReadSparse . . . 22
ReadSparseHardcalls . . . 23
VariantScores . . . 23

Index **25**

pgenlibr-package *PLINK 2 Binary (.pgen) Reader*

Description

A thin wrapper over PLINK 2's core libraries which provides an R interface for reading .pgen files. A minimal .pvar loader and a basic .pgen writer are also included.

Details

`NewPvar` and `NewPgen` initialize the respective readers. Then, you can either iterate through one variant at a time (`Read`, `ReadAlleles`) or perform a multi-variant matrix load (`ReadIntList`, `ReadList`). When you're done, `ClosePgen` and `ClosePvar` free resources.

To write a .pgen, call `NewPgenWriter`, append one variant at a time (`AppendBiallelic`, `AppendAlleles`, `AppendDosages`), and then call `ClosePgenWriter`. Only the .pgen is written; the companion .pvar and .psam files must be written separately.

Author(s)

Christopher Chang <chrchang@alumni.caltech.edu>

References

Chang, C.C. and Chow, C.C. and Tellier, L.C.A.M. and Vattikuti, S. and Purcell, S.M. and Lee J.J. (2015) Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigascience* 4:7. doi:10.1186/s1374201500478.

Examples

```
# This is modified from https://yosuketanigawa.com/posts/2020/09/PLINK2 .
library(pgenlibr)

# These files are subsetted from downloads available at
#   https://www.cog-genomics.org/plink/2.0/resources#phase3_1kg .
# Note that, after downloading the original files, the .pgen file must be
# decompressed before use; but both pgenlibr and the PLINK 2 program can
# handle compressed .pvar files.
pvar_path <- system.file("extdata", "chr21_phase3_start.pvar.zst", package="pgenlibr")
pgen_path <- system.file("extdata", "chr21_phase3_start.pgen", package="pgenlibr")

pvar <- pgenlibr::NewPvar(pvar_path)
pgen <- pgenlibr::NewPgen(pgen_path, pvar=pvar)

# Check the number of variants and samples.
pgenlibr::GetVariantCt(pgen)
pgenlibr::GetRawSampleCt(pgen)

# Get the chromosome, position, and ID of the first variant.
GetVariantChrom(pvar, 1)
```

```
GetVariantPos(pvar, 1)
GetVariantId(pvar, 1)

# Read the 14th variant.
buf <- pgenlibr::Buf(pgen)
pgenlibr::Read(pgen, buf, 14)

# Get the index of the variant with ID "rs569225703".
var_id <- pgenlibr::GetVariantsById(pvar, "rs569225703")

# Get allele count.
pgenlibr::GetAlleleCt(pvar, var_id)

# It has three alleles, i.e. two ALT alleles.
# Read first-ALT-allele dosages for that variant.
pgenlibr::Read(pgen, buf, var_id)

# Read second-ALT-allele dosages.
pgenlibr::Read(pgen, buf, var_id, allele_num=3)

# Read a matrix with both variants.  Note that, for the multiallelic variant,
# the dosages of both ALT alleles are summed here.
geno_mat <- pgenlibr::ReadList(pgen, c(14, var_id))

pgenlibr::ClosePgen(pgen)
pgenlibr::ClosePvar(pvar)
```

`AlleleCodeBuf` *Returns an empty two-row numeric matrix that ReadAlleles() can load to.*

Description

Returns an empty two-row numeric matrix that ReadAlleles() can load to.

Usage

```
AlleleCodeBuf(pgen)
```

Arguments

`pgen`	Object returned by NewPgen().

Value

Numeric matrix with two rows, and appropriate number of columns for ReadAlleles().

AppendAlleles *Appends a hardcall variant, specified as allele codes, to a .pgen being written. Unlike AppendBiallelic(), this supports phased and multiallelic hardcalls.*

Description

Appends a hardcall variant, specified as allele codes, to a .pgen being written. Unlike AppendBiallelic(), this supports phased and multiallelic hardcalls.

Usage

```
AppendAlleles(
  pgen_writer,
  acbuf,
  phasepresent = NULL,
  all_phased = FALSE,
  allele_ct = NULL
)
```

Arguments

pgen_writer	Object returned by NewPgenWriter().
acbuf	Integer or numeric matrix with 2 rows and sample_ct columns, in the format filled by ReadAlleles(): 0 corresponds to the REF allele, 1 to the first ALT, 2 to the second ALT, etc., and a missing hardcall is represented by a pair of NA (or -9) codes. For a phased heterozygous call, the first row holds the allele on the first haplotype; unphased heterozygous calls can be stored in either order.
phasepresent	Logical vector of length sample_ct indicating which calls are phased; entries for non-heterozygous calls are ignored. Optional. (A buffer filled by ReadAlleles() is in this format.) Requires hardcall_phase_present = TRUE in the NewPgenWriter() call.
all_phased	Whether to treat all calls as phased; defaults to FALSE. Cannot be combined with phasepresent, and requires hardcall_phase_present = TRUE in the NewPgenWriter() call. If neither phasepresent nor all_phased is specified, all calls are unphased.
allele_ct	Number of alleles (REF + ALTs) for this variant. Optional; by default, this is inferred to be max(2, 1 + <max observed allele code>). It must be specified when the last ALT allele in the .pvar is unobserved.

Value

No return value, called for side-effect.

`AppendBiallelic` *Appends an unphased biallelic hardcall variant to a .pgen being written.*

Description

Appends an unphased biallelic hardcall variant to a .pgen being written.

Usage

```
AppendBiallelic(pgen_writer, geno)
```

Arguments

`pgen_writer`	Object returned by NewPgenWriter().
`geno`	Integer or numeric vector of length sample_ct, with values in {0, 1, 2, NA} indicating the number of ALT allele copies; -9 is also accepted as a missing-value code. (A buffer filled by ReadHardcalls() is in this format.)

Value

No return value, called for side-effect.

`AppendDosages` *Appends a biallelic dosage variant to a .pgen being written.*

Description

Dosages are stored with 1/16384 precision. Hardcalls are saved as well: each is the nearest integer when the dosage is within 0.1 of it, and missing otherwise.

Usage

```
AppendDosages(pgen_writer, dosages)
```

Arguments

`pgen_writer`	Object returned by NewPgenWriter(), with dosage_present = TRUE.
`dosages`	Numeric vector of length sample_ct, with ALT allele dosages in [0, 2], or NA for missing. (A buffer filled by Read() is in this format.)

Value

No return value, called for side-effect.

`BoolBuf` *Returns a bool buffer that ReadAlleles() can load phasing information to.*

Description

Returns a bool buffer that ReadAlleles() can load phasing information to.

Usage

```
BoolBuf(pgen)
```

Arguments

`pgen`	Object returned by NewPgen().

Value

Logical vector with appropriate length for ReadAlleles().

`Buf` *Returns a numeric buffer that Read() or ReadHardcalls() can load to.*

Description

Returns a numeric buffer that Read() or ReadHardcalls() can load to.

Usage

```
Buf(pgen)
```

Arguments

`pgen`	Object returned by NewPgen().

Value

Numeric vector with appropriate length for Read() and ReadHardcalls().

`ClosePgen` *Closes a pgen object, releasing resources.*

Description

Closes a pgen object, releasing resources.

Usage

```
ClosePgen(pgen)
```

Arguments

`pgen`	Object returned by NewPgen().

Value

No return value, called for side-effect.

`ClosePgenWriter` *Finalizes a .pgen being written, and releases resources. This must be called after all variant_ct variants have been appended; the .pgen is not valid before then.*

Description

Finalizes a .pgen being written, and releases resources. This must be called after all variant_ct variants have been appended; the .pgen is not valid before then.

Usage

```
ClosePgenWriter(pgen_writer)
```

Arguments

`pgen_writer`	Object returned by NewPgenWriter().

Value

No return value, called for side-effect.

`ClosePvar` *Closes a pvar object, releasing memory.*

Description

Closes a pvar object, releasing memory.

Usage

```
ClosePvar(pvar)
```

Arguments

`pvar`	Object returned by NewPvar().

Value

No return value, called for side-effect.

`GetAlleleCode` *Look up an allele code.*

Description

Look up an allele code.

Usage

```
GetAlleleCode(pvar, variant_num, allele_num)
```

Arguments

`pvar`	Object returned by NewPvar().
`variant_num`	Variant index (1-based).
`allele_num`	Allele index (1-based).

Value

The allele_numth allele code for the variant_numth variant. allele_num=1 corresponds to the REF allele, allele_num=2 corresponds to the first ALT allele, allele_num=3 corresponds to the second ALT allele if it exists and errors out otherwise, etc.

`GetAlleleCt` *Returns the effective number of alleles for a variant. Note that if no pvar was provided to the NewPgen() call, this function may return 2 even at multiallelic variants, since the .pgen may not store allele-count information.*

Description

Returns the effective number of alleles for a variant. Note that if no pvar was provided to the NewPgen() call, this function may return 2 even at multiallelic variants, since the .pgen may not store allele-count information.

Usage

```
GetAlleleCt(pvar_or_pgen, variant_num)
```

Arguments

`pvar_or_pgen`	Object returned by NewPvar() or NewPgen().
`variant_num`	Variant index (1-based).

Value

max(2, <number of alleles the variant_numth variant is known to have>). Note that if no

`GetMaxAlleleCt` *Returns the maximum GetAlleleCt() value across all variants in the file.*

Description

Returns the maximum GetAlleleCt() value across all variants in the file.

Usage

```
GetMaxAlleleCt(pvar_or_pgen)
```

Arguments

`pvar_or_pgen`	Object returned by NewPvar() or NewPgen().

Value

Maximum GetAlleleCt() value across all variants.

GetRawSampleCt *Returns the number of samples in the file.*

Description

Returns the number of samples in the file.

Usage

```
GetRawSampleCt(pgen)
```

Arguments

pgen	Object returned by NewPgen().

Value

Number of samples.

GetVariantChrom *Retrieve chromosome ID for given variant index.*

Description

Retrieve chromosome ID for given variant index.

Usage

```
GetVariantChrom(pvar, variant_num)
```

Arguments

pvar	Object returned by NewPvar().
variant_num	Variant index (1-based).

Value

Chromosome ID for the variant_numth variant.

GetVariantCt *Returns the number of variants in the file.*

Description

Returns the number of variants in the file.

Usage

```
GetVariantCt(pvar_or_pgen)
```

Arguments

pvar_or_pgen	Object returned by NewPvar() or NewPgen().

Value

Number of variants.

GetVariantId *Convert variant index to variant ID string.*

Description

Convert variant index to variant ID string.

Usage

```
GetVariantId(pvar, variant_num)
```

Arguments

pvar	Object returned by NewPvar().
variant_num	Variant index (1-based).

Value

The variant_numth variant ID string.

`GetVariantPos` *Retrieve POS (base-pair coordinate on a chromosome) for given variant index.*

Description

Retrieve POS (base-pair coordinate on a chromosome) for given variant index.

Usage

```
GetVariantPos(pvar, variant_num)
```

Arguments

`pvar`	Object returned by NewPvar().
`variant_num`	Variant index (1-based).

Value

POS for the variant_numth variant.

`GetVariantsById` *Convert variant ID string to variant index(es).*

Description

Convert variant ID string to variant index(es).

Usage

```
GetVariantsById(pvar, id)
```

Arguments

`pvar`	Object returned by NewPvar().
`id`	Variant ID to look up.

Value

A list of all (1-based) variant indices with the given variant ID.

`GetWrittenVariantCt` *Returns the number of variants appended so far to a .pgen being written.*

Description

Returns the number of variants appended so far to a .pgen being written.

Usage

```
GetWrittenVariantCt(pgen_writer)
```

Arguments

`pgen_writer`	Object returned by NewPgenWriter().

Value

Number of variants appended so far.

`HardcallPhasePresent` *Returns whether explicitly phased hardcalls are present.*

Description

Returns whether explicitly phased hardcalls are present.

Usage

```
HardcallPhasePresent(pgen)
```

Arguments

`pgen`	Object returned by NewPgen().

Value

TRUE if the file contains at least one phased heterozygous hardcall, FALSE otherwise.

HasSparse *Returns whether dosages for the variant_numth variant and given allele are represented in a sparse manner that is supported by ReadSparse(), under the current sample subset.*

Description

Returns whether dosages for the variant_numth variant and given allele are represented in a sparse manner that is supported by ReadSparse(), under the current sample subset.

Usage

```
HasSparse(pgen, variant_num, allele_num = 2L)
```

Arguments

pgen	Object returned by NewPgen().
variant_num	Variant index (1-based).
allele_num	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults to 2.

Value

True iff the (variant, allele) pair has a sparse representation that can be returned by ReadSparse().

HasSparseHardcalls *Returns whether hardcalls for the variant_numth variant and given allele are represented in a sparse manner that is supported by ReadSparseHardcalls().*

Description

Returns whether hardcalls for the variant_numth variant and given allele are represented in a sparse manner that is supported by ReadSparseHardcalls().

Usage

```
HasSparseHardcalls(pgen, variant_num, allele_num = 2L)
```

Arguments

pgen	Object returned by NewPgen().
variant_num	Variant index (1-based).
allele_num	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults to 2.

Value

True iff the (variant, allele) pair has a sparse representation that can be returned by ReadSparseHardcalls().

IntAlleleCodeBuf *Returns an empty two-row integer matrix that ReadAlleles() can load to.*

Description

Returns an empty two-row integer matrix that ReadAlleles() can load to.

Usage

```
IntAlleleCodeBuf(pgen)
```

Arguments

pgen	Object returned by NewPgen().

Value

Integer matrix with two rows, and appropriate number of columns for ReadAlleles().

IntBuf *Returns an integer buffer that ReadHardcalls() can load to.*

Description

Returns an integer buffer that ReadHardcalls() can load to.

Usage

```
IntBuf(pgen)
```

Arguments

pgen	Object returned by NewPgen().

Value

Integer vector with appropriate length for ReadHardcalls().

NewPgen *Opens a .pgen or PLINK 1 .bed file.*

Description

Opens a .pgen or PLINK 1 .bed file.

Usage

```
NewPgen(filename, pvar = NULL, raw_sample_ct = NULL, sample_subset = NULL)
```

Arguments

filename	.pgen/.bed file path.
pvar	Object (see NewPvar()) corresponding to the .pgen's companion .pvar; technically optional, but necessary for some functionality. In particular, at multiallelic variants, all ALT alleles may be collapsed together when .pvar information is not available.
raw_sample_ct	Number of samples in file; required if it's a PLINK 1 .bed file, otherwise optional.
sample_subset	List of 1-based positions of samples to load; optional, all samples are loaded if this is not specified.

Value

A pgen object, which can be queried for genotype/dosage data.

NewPgenWriter *Creates a new .pgen file for writing.*

Description

Only the .pgen is written; you are responsible for writing the companion .pvar (or .bim) and .psam (or .fam) files yourself, with samples in the same order as in the vectors passed to the append functions, and variants in append order. After all variant_ct variants have been appended (with AppendBiallelic(), AppendAlleles(), or AppendDosages()), ClosePgenWriter() must be called to finalize the file.

Usage

```
NewPgenWriter(
  filename,
  sample_ct,
  variant_ct,
  nonref_flags = as.logical(c(TRUE)),
  allele_ct_limit = 2L,
  hardcall_phase_present = FALSE,
  dosage_present = FALSE
)
```

Arguments

filename	.pgen file path.
sample_ct	Number of samples.
variant_ct	Number of variants that will be written.
nonref_flags	TRUE when the data is from an ordinary PLINK 1 .bed file (where the A2 allele is major rather than consistently reference), FALSE when the REF allele is consistently reference, or a logical vector of length variant_ct when this is mixed. NULL is also permitted (this delegates tracking of nonref information to the .pvar file), but it's discouraged since .pgen+.bim+.fam is a useful data representation with direct plink2 support. Defaults to TRUE, as in the Python PgenWriter.
allele_ct_limit	Maximum number of alleles (REF + ALTs) that any variant can have; defaults to 2. Values larger than 2 are only useful with AppendAlleles().
hardcall_phase_present	Whether phased hardcalls may be written (with AppendAlleles()). Defaults to FALSE.
dosage_present	Whether dosages may be written (with AppendDosages()). Defaults to FALSE.

Value

A pgen_writer object.

NewPvar *Loads variant positions, IDs, and allele codes from a .pvar or .bim file (which can be compressed with gzip or Zstd).*

Description

Loads variant positions, IDs, and allele codes from a .pvar or .bim file (which can be compressed with gzip or Zstd).

Usage

```
NewPvar(filename, omit_chrom = FALSE, omit_pos = FALSE)
```

Arguments

filename	.pvar/.bim file path.
omit_chrom	Whether to skip CHROM column.
omit_pos	Whether to skip POS column.

Value

A pvar object, which can be queried for variant IDs and allele codes.

Read — *Loads the variant_numth variant, and then fills buf with numeric dosages in [0, 2] indicating the dosages of the first ALT (or user-specified) allele for each sample, with missing values represented by NA.*

Description

This function treats the data as diploid; divide by 2 to obtain haploid dosages.

Usage

```
Read(pgen, buf, variant_num, allele_num = 2L)
```

Arguments

pgen	Object returned by NewPgen().
buf	Buffer returned by Buf().
variant_num	Variant index (1-based).
allele_num	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults to 2.

Value

No return value, called for buf-filling side-effect.

`ReadAlleles` *Loads the variant_numth variant, and then fills acbuf with integer allele codes, where each column of the buffer corresponds to a sample. An allele code of 0 corresponds to the REF allele, 1 to the first ALT, 2 to the second ALT, etc. Missing hardcalls are represented by a pair of NA codes.*

Description

This function treats the data as diploid. If it's really haploid, you may want to compare the two rows, and then treat samples where the allele codes differ as missing values.

Usage

```
ReadAlleles(pgen, acbuf, variant_num, phasepresent_buf = NULL)
```

Arguments

`pgen`	Object returned by NewPgen().
`acbuf`	Buffer returned by AlleleCodeBuf() or IntAlleleCodeBuf().
`variant_num`	Variant index (1-based).
`phasepresent_buf`	Buffer returned by BoolBuf(). Optional; if provided, elements are set to true when the sample has known phase. Most of these values will be TRUE even when the raw data is unphased, because homozygous genotypes always have known phase. (Missing genotypes are considered to have unknown phase.)

Value

No return value, called for acbuf-filling side-effect.

`ReadHardcalls` *Loads the variant_numth variant, and then fills buf with {0, 1, 2, NA} values indicating the number of copies of the first ALT (or user-specified) allele each sample has.*

Description

This function treats the data as diploid; you can divide by 2, and then treat 0.5 as NA, if it's actually haploid.

Usage

```
ReadHardcalls(pgen, buf, variant_num, allele_num = 2L)
```

Arguments

pgen	Object returned by NewPgen().
buf	Buffer returned by Buf() or IntBuf().
variant_num	Variant index (1-based).
allele_num	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults to 2.

Value

No return value, called for buf-filling side-effect.

ReadIntList *Load hardcalls for multiple variants as an integer matrix.*

Description

This function treats the data as diploid; you can divide by 2, and then treat 0.5 as NA, if it's actually haploid.

Usage

```
ReadIntList(pgen, variant_subset)
```

Arguments

pgen	Object returned by NewPgen().
variant_subset	Integer vector containing 1-based indexes of variants to load.

Value

Integer matrix, where rows correspond to samples, columns correspond to variant_subset, and values are in {0, 1, 2, NA} indicating the number of hardcall ALT allele copies. For multiallelic variants, all ALT alleles are combined.

`ReadList` *Load dosages for multiple variants as a numeric matrix.*

Description

This function treats the data as diploid; divide by 2 to obtain haploid dosages.

Usage

```
ReadList(pgen, variant_subset, meanimpute = FALSE)
```

Arguments

`pgen`	Object returned by NewPgen().
`variant_subset`	Integer vector containing 1-based indexes of variants to load.
`meanimpute`	Optional; if true, missing values are mean-imputed instead of being represented by NA.

Value

Numeric matrix, where rows correspond to samples, and columns correspond to variant_subset. Values are in [0, 2] indicating ALT allele dosages, or NA for missing dosages. For multiallelic variants, all ALT alelles are combined.

`ReadSparse` *If HasSparse() is true, returns a sparse representation for the (variant, allele) pair. If HasSparse() is false, the function fails.*

Description

If HasSparse() is true, returns a sparse representation for the (variant, allele) pair. If HasSparse() is false, the function fails.

Usage

```
ReadSparse(pgen, variant_num, allele_num = 2L)
```

Arguments

`pgen`	Object returned by NewPgen().
`variant_num`	Variant index (1-based).
`allele_num`	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults to 2.

Value

An object where "sample_nums" is an increasing sequence of positive integers listing which samples have the allele, and "dosages" is a vector listing the dosages (on a 0-2 scale) for those samples.

`ReadSparseHardcalls` *If HasSparseHardcalls() is true, returns a sparse representation for the (variant, allele) pair. If HasSparseHardcalls() is false, the function fails.*

Description

If HasSparseHardcalls() is true, returns a sparse representation for the (variant, allele) pair. If HasSparseHardcalls() is false, the function fails.

Usage

```
ReadSparseHardcalls(pgen, variant_num, allele_num = 2L, return_ints = FALSE)
```

Arguments

`pgen`	Object returned by NewPgen().
`variant_num`	Variant index (1-based).
`allele_num`	Allele index; 1 corresponds to REF, 2 to the first ALT allele, 3 to the second ALT allele if it exists, etc. Optional, defaults to 2.
`return_ints`	Whether to make the "counts" component of the return value an IntegerVector instead of a NumericVector; defaults to false.

Value

An object where "sample_nums" is an increasing sequence of positive integers listing which samples have the allele, and "counts" is a vector listing the allele counts for those samples.

`VariantScores` *Compute variant scores.*

Description

This function treats the data as diploid; divide by 2 to obtain scores based on a haploid dosage matrix.

Usage

```
VariantScores(pgen, weights, variant_subset = NULL)
```

Arguments

`pgen`	Object returned by NewPgen().
`weights`	Sample weights.
`variant_subset`	Integer vector containing 1-based indexes of variants to include in the dosage matrix. Optional; by default, all variants are included.

Value

Numeric vector, containing product of sample-weight vector and the specified subset of the dosage matrix.

Index

∗ **package**
pgenlibr-package, 3

AlleleCodeBuf, 4
AppendAlleles, 5
AppendBiallelic, 6
AppendDosages, 6

BoolBuf, 7
Buf, 7

ClosePgen, 8
ClosePgenWriter, 8
ClosePvar, 9

GetAlleleCode, 9
GetAlleleCt, 10
GetMaxAlleleCt, 10
GetRawSampleCt, 11
GetVariantChrom, 11
GetVariantCt, 12
GetVariantId, 12
GetVariantPos, 13
GetVariantsById, 13
GetWrittenVariantCt, 14

HardcallPhasePresent, 14
HasSparse, 15
HasSparseHardcalls, 15

IntAlleleCodeBuf, 16
IntBuf, 16

NewPgen, 17
NewPgenWriter, 17
NewPvar, 18

pgenlibr *(*pgenlibr-package*)*, 3
pgenlibr-package, 3

Read, 19
ReadAlleles, 20
ReadHardcalls, 20
ReadIntList, 21
ReadList, 22
ReadSparse, 22
ReadSparseHardcalls, 23

VariantScores, 23