

# GxE.scan

October 29, 2025

## Overview

GxE.scan can process a GWAS scan using the `snp.logistic`, `additive.test`, `snp.score` or `snp.matched` functions, whereas `snp.scan.logistic` only calls `snp.logistic`. GxE.scan can process different genotype formats by calling PLINK or GLU to automatically transform the genotype data, so that the user does not have to manually transform the data beforehand. GxE.scan streams the genotype data by reading in a single SNP at a time so that there will be no memory issues, and GxE.scan also has the capacity for users to run their own customized scan by writing a simple R function.

## Example of GxE.scan with a TPED genotype file

For these examples, we will call the `snp.logistic` function for each SNP in the genotype data. The genotype data is in a TPED format. Since TPED files do not contain subject ids, we must also specify the `subject.list` option which defines the file that lists the order of the subjects in the genotype data. We can use the corresponding TFAM file for the TPED subject ids. Only the id columns in the TFAM file are used, so that any other column in this file can have all missing values. This file need not be the TFAM file, just a file containing the ordered genotype subject ids that can be matched to ids in the phenotype data. Get the paths to the genotype data and subject file.

```
> geno.file <- system.file("sampleData", "geno_data.tped.gz", package="CGEN")
> subject.file <- system.file("sampleData", "geno_data.tfam", package="CGEN")
> geno.file
```

```
[1] "/private/var/folders/db/4tvngx8jx4z3fm1gz1nlzw9rc0000gq/T/Rtmpw8RdiC/Rinst4be742d35b94"
```

```
> subject.file
```

```
[1] "/private/var/folders/db/4tvngx8jx4z3fm1gz1nlzw9rc0000gq/T/Rtmpw8RdiC/Rinst4be742d35b94"
```

```
> library(CGEN)
```

The `subject.list` option can be defined either way below. For PLINK genotype formats and when `subject.list` is defined in the second statement below, GxE.scan assumes that the first 2 columns of `subject.file` are the id columns.

```
> subject.list <- list(file=subject.file, id.var=1:2, header=0, delimiter=" ")
> subject.list <- subject.file
```

Define the snp.list argument. The genotype file corresponds to format="tped" or file.type=12.

```
> snp.list <- list(file=geno.file, format="tped", subject.list=subject.list)
```

Define the pheno.list argument. This tab-delimited file contains columns "Family" and "Subject" that correspond to the first 2 columns of the TFAM file. Both these column names must be specified in the id.var option so that the subjects in the phenotype file can be correctly matched to the genotype data. The set of subjects used in the analysis will be the subjects in the phenotype file that have matching ids in the genotype data. The response column is called "CaseControl" and the exposure column is called "Exposure".

```
> pheno.file <- system.file("sampleData", "pheno.txt", package="CGEN")
> pheno.list <- list(file=pheno.file, id.var=c("Family", "Subject"),
+                   file.type=3, delimiter="\t",
+                   response.var="CaseControl", strata.var="Study",
+                   main.vars=c("Gender", "Exposure", "Smoke_CURRENT", "Smoke_FORMER"),
+                   int.vars="Exposure")
```

Define the output file and options list.

```
> out.file <- paste(getwd(), "/GxE.out", sep="")
> op <- list(out.file=out.file)
```

Call the scan function and display the first 5 rows of the output file.

```
> GxE.scan(snp.list, pheno.list, op=op)
```

NOTE: The PLINK software was not found

NOTE: The GLU software was not found

```
[1] "/private/var/folders/db/4tvngx8jx4z3fm1gzlnlzw9rc0000gq/T/Rtmpw8RdiC/Rbuild4be734938c1"
```

```
> x <- read.table(out.file, header=1, as.is=TRUE, sep="\t")
> x[1:5, ]
```

	SNP	MajMinAllele	MAF	MissRate	Case.Counts.012	Control.Counts.012
1	rs2875775	A C	0.4462500	0.0000	63 94 45	65 93 40
2	rs799730	A C	0.2762500	0.0000	113 63 26	118 54 26
3	rs7736921	A C	0.1987500	0.0000	139 54 9	132 45 21
4	rs5968544	A C	0.3123393	0.0275	100 67 29	106 56 31
5	rs5986003	A C	0.3652850	0.0350	92 65 37	82 77 33
	UML.Omnibus.Pvalue CML.Omnibus.Pvalue EB.Omnibus.Pvalue UML.Inter.Pvalue					
1		0.6830582		0.5638221	0.5638214	0.4848020
2		0.7055648		0.7982824	0.9275834	0.4735131
3		0.3485956		0.3120194	0.3123157	0.6184467
4		0.6902793		0.5689191	0.7649808	0.4580637
5		0.5279047		0.7554874	0.7190142	0.2879125
	CML.Inter.Pvalue EB.Inter.Pvalue					
1		0.3498157		0.3498190		
2		0.6971841		0.8289900		
3		0.6422573		0.6390692		
4		0.3280528		0.7897851		
5		0.5300096		0.4734673		

## User-defined scan

A customized scan can be run by writing a simple R function that inputs 2 arguments and returns either a named list or named vector. The first input argument is a data frame containing all the data, and the second argument is a list containing the GxE.scan options, pheno.list and possibly a list named "scan.func.op". The data frame will have a variable called "SNP" (by default) which contains the SNP and will be coded as 0-1-2 or NA. The names in the return list or vector will be column names in the output file. When calling GxE.scan for a user-defined scan, the "model" option in the GxE.scan options list must be set to 0 and the "scan.func" option must be set to the name of the user-defined scan function.

In this next example, we will define our own scan function called "myscan". Note in the "myscan" function below that the second argument "lists" is not used, but this function still must have two arguments. The function "myscan" below calls snp.logistic and then computes the CML SNP by exposure interaction odds-ratio and confidence interval. The return object is a list with names "Odds.Ratio" and "OR.95.CI" which will be column names in the output file.

```
> myscan <- function(data, lists) {
+
+   # Call snp.logistic
+   fit <- snp.logistic(data, "CaseControl", "SNP", main.vars=~Gender + Exposure,
+                       int.vars=~Exposure, strata.var="Study")
+
+   # Extract the CML log-odds ratio and standard error
+   temp <- fit$CML
+   beta <- temp$params["SNP:Exposure"]
+   se    <- sqrt(temp$cov["SNP:Exposure", "SNP:Exposure"])
+
+   # Compute the odds-ratio and 95 percent confidence interval
+   or    <- exp(beta)
+   l     <- round(exp(beta - 1.96*se), digits=4)
+   u     <- round(exp(beta + 1.96*se), digits=4)
+   ci    <- paste("(", l, ", ", u, ")", sep="")
+
+   # Return list. The names "Odds.Ratio" and "OR.95.CI" will be column names in the output
+   list(Odds.Ratio=or, OR.95.CI=ci)
+ }
```

Define subject.list, snp.list and pheno.list using data from the TPED example.

```
> geno.file <- system.file("sampleData", "geno_data.tped.gz", package="CGEN")
> subject.file <- system.file("sampleData", "geno_data.tfam", package="CGEN")
> subject.list <- subject.file
> snp.list <- list(file=geno.file, format="tped", subject.list=subject.list)
> pheno.file <- system.file("sampleData", "pheno.txt", package="CGEN")
> pheno.list <- list(file=pheno.file, id.var=c("Family", "Subject"),
+                   file.type=3, delimiter="\t",
```

```
+      response.var="CaseControl", strata.var="Study",
+      main.vars=c("Gender", "Exposure", "Smoke_CURRENT", "Smoke_FORMER"),
+      int.vars="Exposure")
```

Define the options list. Note that model must be set to 0 and scan.func to "myscan". Setting geno.counts to 0 will cause the genotype frequency counts not to be output.

```
> op <- list(out.file=out.file, model=0, scan.func="myscan", geno.counts=0)
```

Run the customized scan and display the first 5 rows of the output file.

```
> GxE.scan(snp.list, pheno.list, op=op)
```

NOTE: The PLINK software was not found

NOTE: The GLU software was not found

```
[1] "/private/var/folders/db/4tvngx8jx4z3fm1gzlnlzw9rc0000gq/T/Rtmpw8RdiC/Rbuild4be734938c1"
```

```
> x <- read.table(out.file, header=1, as.is=TRUE, sep="\t")
```

```
> x[1:5, ]
```

	SNP	MajMinAllele	MAF	MissRate	Odds.Ratio	OR.95.CI
1	rs2875775	A C	0.4462500	0.0000	1.2049343	(0.814, 1.7836)
2	rs799730	A C	0.2762500	0.0000	1.0903581	(0.7076, 1.6801)
3	rs7736921	A C	0.1987500	0.0000	1.1291252	(0.6781, 1.8802)
4	rs5968544	A C	0.3123393	0.0275	0.8087355	(0.5279, 1.2389)
5	rs5986003	A C	0.3652850	0.0350	1.1420765	(0.7537, 1.7305)

## Advanced user-defined scan

Here we will use other features of a user-defined scan: scan.setup.func and scan.func.op. The function scan.setup.func is a function that is called once after the phenotype file is loaded, but before the genotype data is processed. This function can be used to modify the phenotype data, fit a base model without any SNP, or create and save objects to be used in the scan function. The object scan.func.op is a list of objects to be used in the scan function scan.func.

In the function mysetup below, new variables are added to the phenotype data, a base model is fit, and values needed for likelihood ratio tests are saved in the scan.func.op list. This function can return NULL or a list containing any of the names "data", "pheno.list", or "scan.func.op". Since the input data frame, pheno.list and scan.func.op are modified in this function, the return list should contain all 3 of these objects.

```
> mysetup <- function(data, lists) {
+
+   # data      Data frame containing all the data except the genotype data.
+   # lists     A list containing pheno.list and possibly scan.func.op
+
+   pheno.list <- lists$pheno.list
+   op.list    <- lists[["scan.func.op", exact=TRUE]]
+   if (is.null(op.list)) op.list <- list()
+   svar       <- pheno.list$strata.var
```

```

+   svec          <- data[, svar]
+
+   # Add indicator variables for study to data
+   data[, "Study_A"] <- as.integer(svec %in% "A")
+   data[, "Study_B"] <- as.integer(svec %in% "B")
+
+   # Include one of the indeicators as a main effect
+   mvars <- pheno.list$main.vars
+   pheno.list$main.vars <- c(mvars, "Study_A")
+
+   # Create a formula to fit a NULL model
+   str <- paste(pheno.list$main.vars, collapse=" + ", sep="")
+   str <- paste(pheno.list$response.var, " ~ ", str, sep="")
+   form <- as.formula(str)
+
+   # Fit the base model
+   fit <- glm(form, data=data, family=binomial())
+   print(summary(fit))
+
+   # Save the formula string. This will be used in the scan function.
+   op.list$form.str <- str
+
+   # Save the log-likelihood and rank for LRT tests
+   op.list$rank.base <- fit$rank
+   op.list$loglike.base <- (2*fit$rank - fit$aic)/2
+
+   # Return modified lists and data
+   list(data=data, pheno.list=pheno.list, scan.func.op=op.list)
+ }

```

Define the scan function "myscan2". In this function, we will compute a likelihood ratio test using the UML estimates and a Wald test using the EB estimates. This function has the option called MAF.cutoff which is defined in the scan.func.op list and is used to run either an additive or dominant genetic model depending on the MAF of the SNP. Notice that the values "loglike.base" and "rank.base" which were created in the function "mysetup" are used for efficiency: no need to re-run the base model for SNPs with no missing values. The returned object from this function should be a named vector or named list.

```

> myscan2 <- function(data, lists) {
+
+   # data      Data frame containing all the data except the genotype data.
+   # lists     A list containing pheno.list and possibly scan.func.op
+
+   plist <- lists$pheno.list
+   op <- lists$scan.func.op
+
+   # By default, the SNP variable snp.var is called "SNP"
+   snp.var <- plist$snp.var
+   int.vars <- plist$int.vars
+
+ }

```

```

+ # Change genetic.model depending on the MAF
+ snp      <- data[, snp.var]
+ missing  <- is.na(snp)
+ nmiss    <- sum(missing)
+ MAF      <- 0.5*mean(snp, na.rm=TRUE)
+ if (MAF < op$MAF.cutoff) {
+   gmodel <- 1
+   str    <- "Dominant"
+ } else {
+   gmodel <- 0
+   str    <- "Additive"
+ }
+
+ # Fit full model
+ fit <- snp.logistic(data, plist$response.var, snp.var,
+   main.vars=plist$main.vars, int.vars=int.vars,
+   strata.var=plist$strata.var,
+   op=list(genetic.model=gmodel))
+
+ # Get the log-likelihood and rank for likelihood ratio tests.
+ loglike.full <- fit$UML$loglike
+ rank.full    <- length(fit$UML$parms)
+
+ # If the SNP had missing values, refit the base model.
+ if (nmiss) {
+   subset <- !missing
+   form   <- as.formula(op$form.str)
+   fit0   <- glm(form, data=data, family=binomial(), subset=subset)
+   rank.base <- fit0$rank
+   loglike.base <- (2*fit0$rank - fit0$aic)/2
+ } else {
+   # No missing values, use saved values from scan.setup.func
+   rank.base <- op$rank.base
+   loglike.base <- op$loglike.base
+ }
+
+ # Compute the likelihood reatio test and p-value
+ LRT <- 2*(loglike.full - loglike.base)
+ LRT.p <- pchisq(LRT, rank.full-rank.base, lower.tail=FALSE)
+
+ # Compute a Wald test of the main effect of SNP and interaction using the EB estimates.
+ vars <- c(snp.var, paste(snp.var, ":", int.vars, sep=""))
+ temp <- getWaldTest(fit, vars, method="EB")
+ EB.p <- temp$p
+ EB.df <- temp$df
+ EB.t <- temp$test
+
+ # Define the return vector.
+ ret <- c(str, LRT, LRT.p, EB.t, EB.p, EB.df)
+ names(ret) <- c("Genetic.Model", "UML.LRT", "UML.LRT.Pvalue",

```

```
+           "EB.Wald.Test", "EB.Wald.Pvalue", "DF")
+
+   ret
+ }
```

Define the list of options for the user-defined scan function.

```
> myoptions <- list(MAF.cutoff=0.05)
```

Define the list of options for the GxE.scan function. Here we set the option scan.setup.func to "mysetup" and the option scan.func.op to myoptions. Also by setting geno.counts, geno.MAF and geno.missRate to 0, the MAF, missing rate and genotype frequency counts will not appear in the output data set.

```
> op <- list(out.file=out.file, model=0, scan.func="myscan2", scan.setup.func="mysetup",
+           scan.func.op=myoptions, geno.counts=0, geno.MAF=0, geno.missRate=0)
```

Run the scan. The base model summary from function "mysetup" gets printed.

```
> GxE.scan(snp.list, pheno.list, op=op)
```

NOTE: The PLINK software was not found

NOTE: The GLU software was not found

Call:

```
glm(formula = form, family = binomial(), data = data)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	-0.16191	0.24572	-0.659	0.510
Gender	-0.19437	0.20361	-0.955	0.340
Exposure	0.09094	0.20240	0.449	0.653
Smoke_CURRENT	0.15498	0.24080	0.644	0.520
Smoke_FORMER	0.01508	0.24945	0.060	0.952
Study_A	0.32915	0.20361	1.617	0.106

(Dispersion parameter for binomial family taken to be 1)

```
Null deviance: 554.48 on 399 degrees of freedom
Residual deviance: 550.64 on 394 degrees of freedom
AIC: 562.64
```

Number of Fisher Scoring iterations: 3

```
[1] "/private/var/folders/db/4tvngx8jx4z3fm1gzlnlzw9rc0000gq/T/Rtmpw8RdiC/Rbuild4be734938c1"
```

Read in the output file and display the first 5 rows.

```
> x <- read.table(out.file, header=1, as.is=TRUE, sep="\t")
> x[1:5, ]
```

	SNP	MajMinAllele	Genetic.Model	UML.LRT	UML.LRT.Pvalue	EB.Wald.Test
1	rs2875775	A C	Additive	0.8964883	0.6387487	1.2845462
2	rs799730	A C	Additive	0.6705349	0.7151468	0.1067729
3	rs7736921	A C	Additive	2.3759949	0.3048311	2.6114921
4	rs5968544	A C	Additive	0.6058146	0.7386676	0.4229184
5	rs5986003	A C	Additive	1.4127254	0.4934357	0.6391472

  

	EB.Wald.Pvalue	DF
1	0.5260952	2
2	0.9480136	2
3	0.2709703	2
4	0.8094023	2
5	0.7264587	2

## Running GxE.scan on a cluster

Although the GxE.scan function can process an entire scan, it would certainly take a long time if running on a single processor. For users with access to a computing cluster, processing an entire scan can be performed much quicker. The steps to accomplish this are:

1. Call the function GxE.scan.partition to create the job files.
2. Submit the jobs.
3. Call GxE.scan.combine to combine all output files after all the jobs have finished.

After creating the job files by calling GxE.scan.partition, it is usually a good idea to see if the jobs will produce reasonable output by running one of the R program files or by submitting one of the job files before submitting all the jobs in step 2.

## Session Information

```
> sessionInfo()
```

```
R version 4.5.1 Patched (2025-09-10 r88807)
```

```
Platform: x86_64-apple-darwin20
```

```
Running under: macOS Monterey 12.7.6
```

```
Matrix products: default
```

```
BLAS: /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRblas.0.dylib
```

```
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-x86_64/Resources/lib/libRlapack.dylib
```

```
locale:
```

```
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
time zone: America/New_York
```

```
tzcode source: internal
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets  methods   base
```



other attached packages:

```
[1] cluster_2.1.8.1 CGEN_3.46.0      mvtnorm_1.3-3  survival_3.8-3
```

loaded via a namespace (and not attached):

```
[1] compiler_4.5.1 Matrix_1.7-4  tools_4.5.1    splines_4.5.1  grid_4.5.1  
[6] lattice_0.22-7
```