Introduction to **STRU**ctural **M**odeling of Latent Variables for General Pedigree

Yeunjoo E. Song, Catherine M. Stein, Nathan J. Morris

February 27, 2015

1 Getting started

The **strum** package implements the framework for structural equation models for general pedigrees described in Morris et al. (2010). It includes both fitting and simulation of a broad range of latent measurement models and structural equation models with covariates, allowing for a wide variety of models including latent growth curve models. It can handle multilevel models, polygenic random effects and linkage random effects. Traditional structural equation models and confirmatory factor analysis may also be performed. The framework implemented now can only handle quantitative variables.

Assuming that you have the **strum** package installed, you first need to load it:

```
> library(strum)
```

Note that all packages that **strum** depends on will be loaded as well. Once it is loaded, now you can start a **strum** analysis.

2 strum Analysis

A **strum** analysis is performed by calling the main function *strum()* which requires two arguments, an object of **strumModel** class and an object of **strumData** class. The **strumModel** is an S4 class that represents the trait model for your data. The **strumData** is an S4 class that contains the input data. The following illustrates the typical steps for a strum analysis.

1. Construct a strumModel by *createStrumModel()* function.
2. Prepare data using *createStrumData()* function.
3. Run analysis by the function call *strum()*.

2.1 strumModel

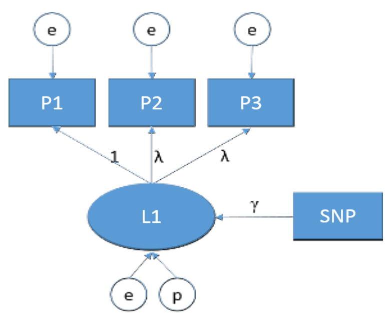
An object of **strumModel** is constructed by *createStrumModel()* function. The *formulas* argument for this function defines the relationship among the variables. The measurement equations are specified by the "=∼" operator. The "∼" operator specifies the structural equations in the model. The "=" operator specifies the constraints in the model, i.e., fixing a model parameter - a variance, covariance, or coefficient. Please refer to the reference manual for more detailed decription of the syntax for the model formulas and other optional arguments for this function.

Following are some acronyms used below:

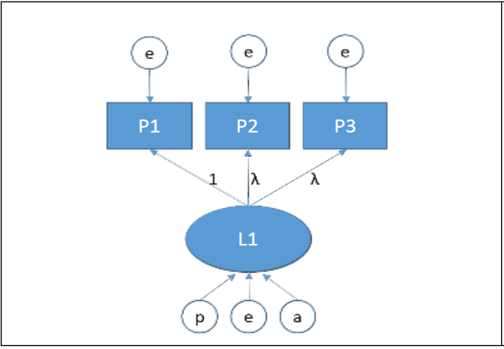
SEM: Structural Equation Model
CFA: Confirmatory Factor Analysis
ACE: Additive polygenic(A), common environmental(C), random error(E)
MIMIC: Multiple Indicators Multiple Causes

Here are some examples of '*formulas*' for different types of analysis with the corresponding model diagrams. Following standard conventions, observed variables are represented as squares and latent variables are represented as ovals. The arrows represent linear relations between variables. There are four possible error terms; additive genetic linkage (a), polygenic (p), shared environmental (c) and independent environmental (e) variance components. They are represented as non-filled small circles. Note that the subscripts for error terms and coefficient parameters are omitted.

- Genetic association analysis model with a latent trait (similar to MIMIC model)

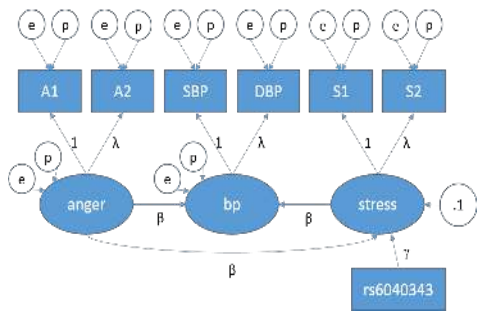
 e
e
e
P1
P2
P3
1
λ
λ
L1
γ
SNP
e
p
	formulas = '
	L1 =∼ P1 + P2 + P3 + <e>
	L1 ∼ SNP + <p,e>
	'

- Genetic linkage analysis model with a latent trait



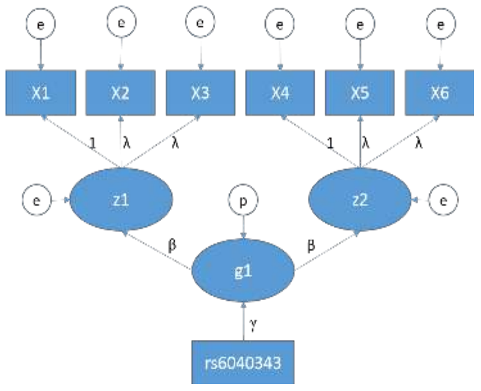
```
formulas = ‘
L1 =~ P1 + P2 + P3 + <e>
L1 ~ <a,p,e>
’
```

- SEM with latent variables and polygenic effect



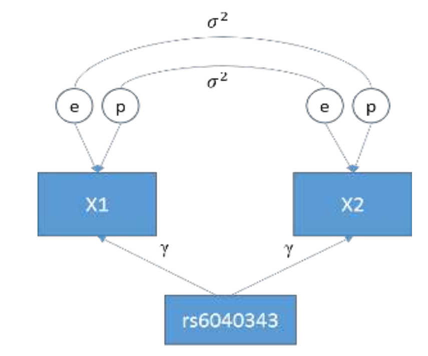
```
formulas = ‘
bp =~ SBP + DBP
anger =~ A1 + A2
stress =~ S1 + S2
bp ~ anger + stress
stress ~ anger + rs6040343
var(stress) = .1
’
```

- CFA with a pleiotropic genetic effect influenced by a SNP



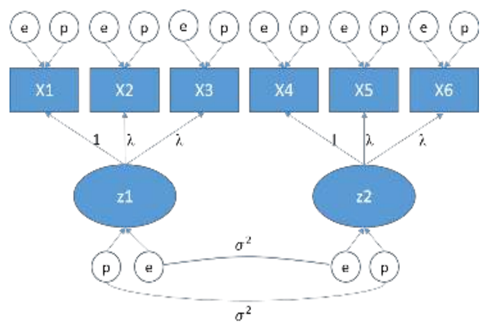
```
formulas = ‘
z1 =~ X1 + X2 + X3 + <e>
z2 =~ X4 + X5 + X6 + <e>
g1 =~ <>
g1 ~ rs6040343 + <p>
z1 ~ g1 + <e>
z2 ~ g1 + <e>
’
```

- Multivariate test for genetic association



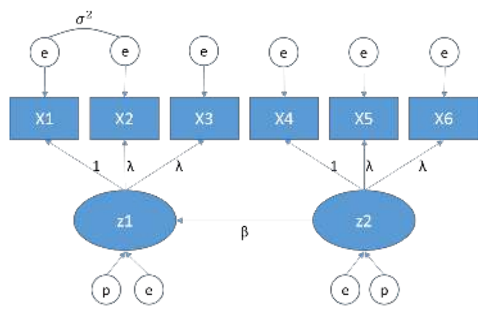
```
formulas = ‘
X1 ~ rs6040343
X2 ~ rs6040343
cov(X1,X2) = NA
’
```

- CFA with polygenic effect



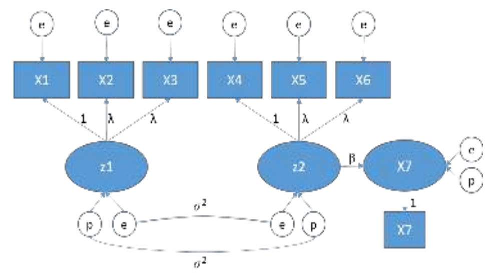
```
formulas = ‘
z1 =~ X1 + X2 + X3
z2 =~ X4 + X5 + X6
cov(z1,z2,p) = NA
cov(z1,z2,e) = NA
’
```

- SEM with latent variables with polygenic effects



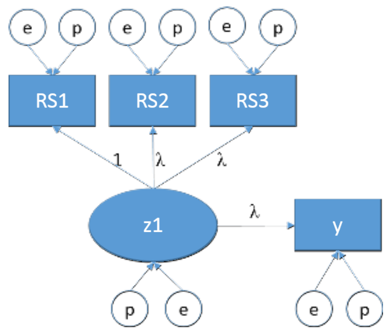
```
formulas = ‘
z1 =~ X1 + X2 + X3 + <e>
z2 =~ X4 + X5 + X6 + <e>
z1 ~ z2
cov(X1,X2,e) = NA
’
```

- CFA with latent variables with polygenic effects



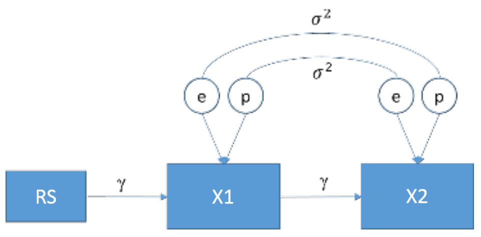
```
formulas = ‘
z1 =~ X1 + X2 + X3 + <e>
z2 =~ X4 + X5 + X6 + <e>
X7 ~ z2
cov(z1,z2) = NA
’
```

- Multiple SNP / Latent Genotype model



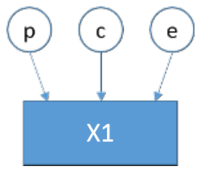
```
formulas = ‘
z1 =~ RS1 + RS2 + RS3 + y
’
```

- Mendelian randomization model



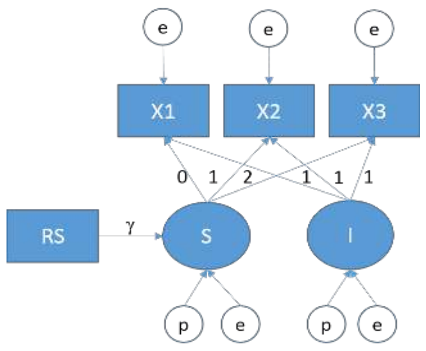
```
formulas = ‘
X2 ~ X1
X1 ~ RS
cov(X1,X2) = NA
’
```

- ACE model - <p, c, e> in strum Model



```
formulas = ‘
X1 ~ <p,c,e>
’
```

- Latent growth curve model



```
formulas = ‘
S =∼ 1*X2 + 2*X3 + <e>
I =∼ 1*X1 + 1*X2 + 1*X3 + <e>
S ∼ RS
’
```

The following example shows how to construct **strumModel** object for the first model above.

```
> formulas =
+   'L1 =~ P1 + P2 + P3 + <e>
+    L1 ~ aSNP + <p,e>
+   '
> myModel = createStrumModel(formulas = formulas)

 Creating strumModel ............................ Done

> myModel

Basic properties of the model:
         Model Class .................. strumModel
         Ascertainment ...................... FALSE

List of all variables:
       Obs Covariate InEita   InY Exogen.
L1   FALSE     FALSE   TRUE FALSE   FALSE
aSNP  TRUE      TRUE  FALSE FALSE      NA
P1    TRUE     FALSE  FALSE  TRUE      NA
P2    TRUE     FALSE  FALSE  TRUE      NA
P3    TRUE     FALSE  FALSE  TRUE      NA

Model formulas:
  L1 =~ P1 + P2 + P3 + <e>
  L1 ~ aSNP + <p,e>
```

2.2 strumData

An object of **strumData** is constructed by *createStrumData()* function. The value of *inData* argument for this function has to be a data.frame, and the allowed values for *dType* argument is either "*Pedigree*" or "*RawData*".

- Pedigree data

  If *dType* = "*Pedigree*", the data must be a data.frame with 4 required id fields - family, id, father, mother. For founders, "0" needs to be used to indicate the missing parents.

  A typical space-delimited pedigree file will look like following.

```
family id father mother sex X1 X2 ...
1 1 0 0 1 0.24 0.36 ...
1 2 0 0 2 1.26 3.25 ...
1 3 1 2 1 0.37 0.48 ...
1 4 1 2 1 3.26 2.67 ...
2 1 0 0 1 2.25 1.87 ...
2 2 0 0 2 0.48 1.68 ...
2 3 1 2 1 1.94 0.62 ...
2 4 1 2 2 3.17 2.10 ...
```

  A typical comma-delimited pedigree file will look like following.

```
family,id,father,mother,sex,X1,X2,...
1,1,0,0,1,0.24,0.36,...
1,2,0,0,2,1.26,3.25,...
1,3,1,2,1,0.37,0.48,...
1,4,1,2,1,3.26,2.67,...
2,1,0,0,1,2.25,1.87,...
2,2,0,0,2,0.48,1.68,...
2,3,1,2,1,1.94,0.62,...
2,4,1,2,2,3.17,2.10,...
```

  The following example show how to construct **strumData** object from an example pedigree file. Note that the first 4 column names are changed to the 4 required id fields - family, id, father, mother.

```
> inPed = system.file("extdata/example_ped.csv", package = "strum")
> dfPed = read.csv(inPed, header=T)[,c(1:6,8:10,17)]
> names(dfPed)[1:4] = c("family","id", "father","mother")
> myPedData = createStrumData(dfPed, "Pedigree")

 Creating strumData ............................. Done
```

```
> myPedData

Data type: Pedigree
Data size: 477 entries, 10 variables

First 5 rows of data values:
  family id father mother sex disease          P1         P2         P3 aSNP
1      1  1      0      0   0       0  0.4093955 0.44450079 -0.3867515    1
2      1  2      0      0   1       0 -1.5037814 1.52582608  0.8832360    0
3      1  3      1      2   0       0  1.5850090 0.08833692  0.9322619    1
4      1  4      1      2   1       0  1.6246356 0.60065352  1.0895325    0
5      1  5      1      2   1       0 -0.4111477 0.08588345 -0.6477336    1

phi object contains  75  matrices:
First matrix:
$`1`
    1   2   3   4   5   6   7
1 1.0 0.0 0.5 0.5 0.5 0.5 0.5
2 0.0 1.0 0.5 0.5 0.5 0.5 0.5
3 0.5 0.5 1.0 0.5 0.5 0.5 0.5
4 0.5 0.5 0.5 1.0 0.5 0.5 0.5
5 0.5 0.5 0.5 0.5 1.0 0.5 0.5
6 0.5 0.5 0.5 0.5 0.5 1.0 0.5
7 0.5 0.5 0.5 0.5 0.5 0.5 1.0

Empty IBD object.
```

- Pedigree data with IBD info

  When your analysis model includes the additive genetic variance component (a), the ibd information for the family data has to be imported by specifying the name of ibd file into *ibdFileName* argument. Currently, the ibd file generated by the program GENIBD in S.A.G.E. package is supported.

```
> iName = system.file("extdata/GENIBD.chr1Ped.ibd", package = "strum")
> myPedDataIBD = createStrumData(dfPed, "Pedigree", ibdFileName=iName)

 Importing S.A.G.E. IBD file ..................... Done
 Creating strumData .............................. Done

> myPedDataIBD
```

```
Data type: Pedigree
Data size: 477 entries, 10 variables

First 5 rows of data values:
  family id father mother sex disease         P1         P2         P3 aSNP
1      1  1      0      0   0       0  0.4093955 0.44450079 -0.3867515    1
2      1  2      0      0   1       0 -1.5037814 1.52582608  0.8832360    0
3      1  3      1      2   0       0  1.5850090 0.08833692  0.9322619    1
4      1  4      1      2   1       0  1.6246356 0.60065352  1.0895325    0
5      1  5      1      2   1       0 -0.4111477 0.08588345 -0.6477336    1

phi object contains  75  matrices:
First matrix:
$`1`
    1   2   3   4   5   6   7
1 1.0 0.0 0.5 0.5 0.5 0.5 0.5
2 0.0 1.0 0.5 0.5 0.5 0.5 0.5
3 0.5 0.5 1.0 0.5 0.5 0.5 0.5
4 0.5 0.5 0.5 1.0 0.5 0.5 0.5
5 0.5 0.5 0.5 0.5 1.0 0.5 0.5
6 0.5 0.5 0.5 0.5 0.5 1.0 0.5
7 0.5 0.5 0.5 0.5 0.5 0.5 1.0

IBD object contains  20  markers:
First 5 rows of markers:
       Marker Position
1 chr1marker1      0.0
2 chr1marker2     10.0
3 chr1marker3     20.0
4 chr1marker4     30.0
5 chr1marker5     40.0
First matrix:
$`1`
    1   2   3   4   5   6   7
1 1.0 0.0 0.5 0.5 0.5 0.5 0.5
2 0.0 1.0 0.5 0.5 0.5 0.5 0.5
3 0.5 0.5 1.0 0.5 0.5 0.5 0.5
4 0.5 0.5 0.5 1.0 0.5 0.5 0.5
5 0.5 0.5 0.5 0.5 1.0 0.0 1.0
6 0.5 0.5 0.5 0.5 0.0 1.0 0.0
7 0.5 0.5 0.5 0.5 1.0 0.0 1.0
```

Note that now, the new **strumData** object, `myPedDataIBD`, includes the IBD information.

- Raw data

  For "*RawData*" type, 4 id fields are not required. The program automatically creates the dummy id fields if any of them are not present in the input file by making each individual as a family of own. Note that the last four columns are automatically included in the following example.

```
> inRaw = system.file("extdata/example_raw.csv", package = "strum")
> dfRaw = read.csv(inRaw, header=T)
> head(dfRaw)

  sex disease     X1     X2     X3    X4    X5      X6 aSNP
1   0       0  3.390  4.180  1.300  1.63  1.90 -0.0613    1
2   1       0 -3.490 -2.720 -0.784 -3.55 -2.89 -2.1300    0
3   0       0  0.499 -3.690 -3.640 -1.10  1.47  0.4280    2
4   1       0  2.990  4.920 -0.372  3.81  2.37  0.4940    1
5   0       0 -5.310 -1.860 -2.420 -2.79 -2.46  0.6770    1
6   1       0 -0.232 -0.195  0.927  1.84  1.38 -0.7670    1

> myRawData = createStrumData(dfRaw, "RawData")

 Creating strumData ............................. Done

> myRawData

Data type: RawData
Data size: 150 entries, 13 variables

First 5 rows of data values:
  sex disease     X1    X2     X3    X4    X5      X6 aSNP family id mother
1   0       0  3.390  4.18  1.300  1.63  1.90 -0.0613    1      1  1      0
2   1       0 -3.490 -2.72 -0.784 -3.55 -2.89 -2.1300    0      2  2      0
3   0       0  0.499 -3.69 -3.640 -1.10  1.47  0.4280    2      3  3      0
4   1       0  2.990  4.92 -0.372  3.81  2.37  0.4940    1      4  4      0
5   0       0 -5.310 -1.86 -2.420 -2.79 -2.46  0.6770    1      5  5      0
  father
1      0
2      0
3      0
4      0
5      0
```

2.3 Run analysis

By the function call *strum()* with two previous objects (**strumModel** and **strumData**) as the arguments, you can run a strum analysis. This is an example with the `myModel` object and the `myPedData` object.

```
> myFitResult = strum(myModel, myPedData)

 Start STRUM analysis ...
   Fitting model step 1 ......................... Done
   Fitting model step 2 ......................... Done
   Testing model fit ............................ Done

Analysis completed!
```

The *strum()* function has an optional third argument, *ibdmarkers*. This argument is to specify the names of the IBD markers when you want to analyze a subset of IBD markers instead of all markers by default. As stated previously, this is the case for the analysis models including the additive genetic variance component (a) and the data including the ibd information.

```
> mNames = c("chr1marker1", "chr1marker2")
> myLinkResult = strum(myLinkModel, myPedIBD, ibdMarkers=mNames)
```

As a result of *strum()* run, an object (or a list of objects in case of analysis with multiple IBD markers) of **strumFittedModel** class, which contains the model description and two result tables. The first table contains the fitted parameter values with standard errors, confidence intervals, and p-values. The second table contains the information on the model fit from four different measures, with the degrees of freedom and p-values.

1. the un-adjusted χ^2 index of fit
2. the mean adjusted χ^2 index of fit
3. the mean and variance adjusted χ^2 index of fit
4. the theoretically corrected χ^2 index of fit

The p-value for the theoretically corrected χ^2 index assumes the χ^2 statistic follows the distribution of a weighted sum of χ^2 random variables. To calculate the p-value, we simulate from this theoretical distribution, and we count the number of simulations which exceed the test statistic.

3 Simulation

This package provides the functions to simulate both trait and marker data. Given a model you want to test, it simulates the data according to the model specification. The following illustrates the typical steps for a strum simulation.

1. Construct a simulation model.
   - Import Hapmap data by *importHapmapData()* function.
   - Construct a strumMarker by *createStrumMarker()* function.
   - Construct a strumSimModel by *createSimModel()* function.
2. Simulate data using *simulateStrumData()* function.
3. Run analysis on the simulated data by the function call *strum()*.

3.1 Simulation model

An object of **strumSimModel** is constructed by *createSimModel()* function. The syntax of the *formulas* argument for this function is similar to the one for the *createStrumModel()* function. However, for the simulation, you include the coefficients for the model parameters, and there are two additional optional arguments, "*tMissingRate*" and "*markerInfo*".

The value for "*tMissingRate*" is a numeric vector to define the missing rate(s) of the simulated trait(s) in the model. The length of vector needs to be equal to the number of observed traits in the model or 1 if one common missing rate is applied to all traits. A value has to be given for the *markerInfo* argument when a marker is included in the simulation model. This must be an object of **strumMarker** class with a data.frame containing hapmap data, which you can import from Hapmap project website.

- Import Hapmap data

  To construct a data.frame containing hapmap data, the function *importHapmapData()* comes in handy. This function imports Hapmap3 Phased data from the wabsite for the specified chromosome number. By default, 'CEU' population is used when the population of your choice is not specified. In the following example, the chromosome 20 of 'CEU' population is imported.

  ```
  > hap20 = importHapmapData(20)
  ```

- Construct strumMarker

  An object of **strumMarker** is constructed by *createStrumMarker()* function. The value of "*hapMapData*" argument for this function must be a data.frame containing hapmap data with three required information fields - rsID, chr, and phys_position. In the following example, every 10th SNPs from the imported hapmap data above are used to construct a **strumMarker** object.

```
> #hap20snp10 = hap20[(1:10)*10,]
> #save(hap20snp10,file="hap20snp10.Rdata")
> #using locally saved copy
> inHap = system.file("extdata/hap20snp10.Rdata", package = "strum")
> load(file=inHap)
> snpStrumMarker = createStrumMarker(hapMapData=hap20snp10)

 Creating strumMarker ........................... Done

> #snpStrumMarker
```

- Construct strumSimModel

  Now, you can construct an object of **strumSimModel** by calling *createSimModel()* function. The following example shows a simulation model using the `snpStrumMarker` and a common missing rate of 0.1 for the simulated traits.

```
> simform =
+   'L1 =~ X1 + 2*X2 + 0.5*X3 + <e>
+    L1 ~ aSNP + <p,e>
+   '
> mySimModel = createSimModel(formulas = simform,
+                             tMissingRate = c(0.1),
+                             markerInfo = snpStrumMarker)

 Creating strumSimModel ......................... Done

> #mySimModel
```

3.2 Simulate data

Given an object of **strumSimModel**, the data is simulated by the function *simulateStrumData()* according to the specified model. There are two optional arguments, "*inData*" and "*N*". Note that either "*inData*" or "*N*", or both must be specified.

An object of **strumData** or a data.frame contaning that input data must be given as the value for "*inData*" argument when a specific format of the input data (either family structures or individuals) is desired. This value is also required when the simulation model contains covariate(s), since covariates are not simulated. When a particular size of the simulated data is desired, a positive number needs to given as the value for "*N*" argument.

The following example shows the case when the value for "*inData*" argument is specified using the `myPedData` from above. Note that all simulated traits (P1, P2, p3) will have the missing rate of 0.1.

```
> mySimData = simulateStrumData(mySimModel, myPedData)
```

```
 Simulating strumData ........................... Done
```

```
> #mySimData
```

This is another example when the value for "*N*" argument is specified, with a simple model without any marker variables.

```
> simform1 = 'z1 =~ X1 + 0.8*X2 + 0.5*X3 + y'
> mySimModel1 = createSimModel(formulas = simform1,
+                              defaultError='<e>')
```

```
 Creating strumSimModel ......................... Done
```

```
> #mySimModel1
> mySimData1 = simulateStrumData(mySimModel1, N=150)
```

```
 Creating strumData ............................. Done
 Simulating strumData ........................... Done
```

```
> #mySimData1
```

3.3 Run analysis with simulated data

As for the strum analysis, you can run a strum analysis by the function call *strum()* with simulate data. This is an example with the `mySimData1` data.

```
> testform = 'z1 =~ X1 + X2 + X3 + y'
> myTestModel = createStrumModel(formulas = testform, defaultError='<e>')
> mySimResult = strum(myTestModel, mySimData1)
> #mySimResult
```

4 Ascertainment

This section explains how to specify the ascertainment of the data, a feature commonly found in family-based studies. That is, pedigrees are selected for inclusion in a study based upon some criteria for the observed traits. Generally, pedigrees are collected by first identifying an individual who is affected or has an extreme phenotype. Such an individual, who causes the pedigree to be sampled, is known as a proband.

There is an optional argument, "*ascertainment*", for both "*createStrumModel()*" and "*createSimModel()*" functions. Though the names are the same for both functions, the values are different.

1. The value for *createStrumModel()*

   Charactor stating the name of the column in data file that contains the indicator variable (1, 0) designating the probands of the pedigrees.

   Suppose that "proband" field in the "example.ped" file is a (1, 0) indicator of proband status for this family data. Then, in the association analysis model above, you would specify the *ascertainment* argument value as "proband" for *createStrumModel()* call as below.

```
> myAStrumModel = createStrumModel(formulas = formulas,
+                                   ascertainment="proband")

 Creating strumModel ............................ Done

> myAStrumModel

Basic properties of the model:
         Model Class .................. strumModel
         Ascertainment ...................... TRUE

List of all variables:
       Obs Covariate InEita   InY Exogen.
L1   FALSE     FALSE   TRUE FALSE   FALSE
aSNP  TRUE      TRUE  FALSE FALSE      NA
P1    TRUE     FALSE  FALSE  TRUE      NA
P2    TRUE     FALSE  FALSE  TRUE      NA
P3    TRUE     FALSE  FALSE  TRUE      NA

Model formulas:
  L1 =~ P1 + P2 + P3 + <e>
  L1 ~ aSNP + <p,e>
```

   Note that now, the description of the model informs that this model includes the ascertainment.

2. The value for *createSimModel()*

   Function stating the ascertainment criteria of the data. The return value of the function is composed of two components for each pedigree, allowing either one of components or both components to be returned as a list. The first component is a TRUE/FALSE value indicating the ascertainment status of the pedigree. The second component is a vector of TRUE/FALSE stating the proband status of each member of the pedigree Please refer to the reference manual for details.

   This is an example when the data is simulated with the ascertainment using the 'disease' field in `myPedData` above.

```
> aFunction = function(thisFam)
+             {
+               aff = (thisFam$disease == 1)
+               ascertained = any(aff)
+               proband = rep(FALSE, nrow(thisFam))
+               if(ascertained)
+                 pPos = which.min(thisFam$disease == 1)
+                 proband[pPos] = TRUE
+               return(list(aStatus=ascertained, pStatus=proband))
+             }
> myASimModel = createSimModel(formulas = simform,
+                              markerInfo = snpStrumMarker,
+                              ascertainment = aFunction)

 Creating strumSimModel ......................... Done

> #myASimModel
> myASimData = simulateStrumData(myASimModel, myPedData)

 Simulating strumData ........................... Done

> #myASimData
```

5 SessionInfo

```
> sessionInfo();

R version 3.1.2 (2014-10-31)
Platform: x86_64-w64-mingw32/x64 (64-bit)

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] strum_0.6        Rgraphviz_2.10.0 graph_1.44.1     pedigree_1.4
[5] reshape_0.8.5    HaploSim_1.8.4   Matrix_1.1-4

loaded via a namespace (and not attached):
[1] BiocGenerics_0.12.1 lattice_0.20-29     MASS_7.3-35
[4] parallel_3.1.2      plyr_1.8.1          Rcpp_0.11.3
[7] stats4_3.1.2        tools_3.1.2
```

References

Morris, N.J., Elston, R.C., & Stein, C.M. (2010). A framework for structural equation models in general pedigrees. *Human heredity*, 70, 278–286.