The Global Test
and the *globaltest* R package

Jelle Goeman Jan Oosting Livio Finos

April 22, 2010

Contents

Chapter 1

Introduction

This vignette explains the use of the *globaltest* package. Chapter 2 describes the use of the test and the package from a general statistical perspective. Later chapters explain how to use the *globaltest* package for specific applications.

1.1 Citing *globaltest*

When using the *globaltest* package, please cite one or more of the following papers, as appropriate.

- Goeman et al. (2004) is the original paper describing the global test for linear and logistic regression, and its application to gene set testing.
- Goeman et al. (2005) extends the global test to survival data and explains how to deal with nuisance (null) covariates.
- Goeman et al. (2006) proves the local optimality of the global test and explores its general theoretical properties. This is the core paper of the global test methodology
- Goeman and Mansmann (2008) develops the Focus Level method for multiple testing correction in the Gene Ontology graph
- Goeman et al. (2009) derives the asymptotic distribution of the global test for generalized linear models

1.2 Package overview

The global test is meant for data sets in which many covariates (or features) have been measured for the same subjects, together with a response variable, e.g. a class label, a survival time or a continuous measurement. The global test can be used on a group (or subset) of the covariates, testing whether that group of covariates is associated with the response variable.

The null hypothesis of the global test is that none of the covariates in the tested group is associated with the response. The alternative is that at least one of the covariates has such an association. However, the global test is designed in such a way that it is especially directed against the alternative that most of the covariates are associated with the response in a small way. In fact, against such an alternative the global test is the optimal test to use (Goeman et al., 2006).

The global test is based on regression models in which the distribution of the response variable is modeled as a function of the covariates. The type of regression model depends on the response. Currently implemented models are

- linear regression (continuous response),
- logistic regression (binary response),
- multinomial logistic regression (multi-class response),
- Poisson regression (count response),
- the Cox proportional hazards model (survival response).

Modeling in terms of a regression model makes it easy to adjust the test for the confounding effect of nuisance covariates: covariates that are known to have an effect on the response and which are correlated with (some of) the covariates of interest, and which may, if not adjusted for, lead to spurious associations.

The *globaltest* package implements the global test along with additional functionality. Several diagnostic plots can be used to visualize the test result and to decompose it to see the influence of individual covariates and subjects. Multiple testing procedures are offered for the situation in which a user wants to perform many global tests on the same data, e.g. when testing many alternative subsets. In that case, possible relationships between the test results arise due to subset relationships among tested sets which may be exploited.

The package also offers some functions that are tailored to specific applications of the global test. In the current version, the only application supported in this way is gene set testing (see Chapter 3). Tailored functions for other applications (goodness-of-fit testing, prediction/classification pre-testing, testing for the presence of a random effect) are under development.

1.3 Comparison with the likelihood ratio test

In its most general form, the global test is a score test for nested parametric models, and as such it is a competitor of the likelihood ratio test. It can be used in every situation in which a likelihood ratio test may also be used, but the global test's properties are different from those of the likelihood ratio test. We summarize the differences briefly from a theoretical statistical perspective. For more details, see Goeman et al. (2006).

It is well known that the likelihood ratio test is invariant to the parametrization of the alternative model. The global test does not have this property: it depends on the model's precise parametrization. Therefore, there is not a single global test for a

given pair of null and alternative hypothesis, but a multitude of tests: one for each possible parametrization of the alternative hypothesis. In return for giving up this parametrization-invariance, the global test gains an optimality-property that depends on the parametrization of the model. As detailed in Goeman et al. (2006), the global test is optimal (among all possible tests) on average in a neighborhood of the null hypothesis. The shape of this neighborhood is determined by the parametrization of the alternative hypothesis. In practice, this means that in situations in which a "natural" parametrization of the alternative model exists, the global test for that parametrization is often more powerful than the likelihood ratio test (examples in Goeman et al., 2006).

A second important property of the global test is that it may still be used in situations in which the alternative model cannot be fitted to the data, which may happen, for example, if the alternative model is overparameterized, or in high dimensional situations in which there are more parameters than observations. In such cases the likelihood ratio test usually breaks down, but the global test still functions, often with good power.

Being a score test, the global test is most focused on alternatives close to the null hypothesis. This means that the global test is good at detecting alternatives that have many small effects (in terms of the chosen parametrization), but that it may not be the optimal test to use if the effects are very large.

Chapter 2

The global test

2.1 Global test basics

We illustrate most of the features of the *globaltest* package and its functions with a very simple application on simulated data using a linear regression model. More extensive real examples relating to specific areas of application can be found in later chapters of this vignette.

2.1.1 Example data

We simulate some data

```
> set.seed(1)
> Y <- rnorm(20)
> X <- matrix(rnorm(200), 20, 10)
> X[, 1:3] <- X[, 1:3] + Y
> colnames(X) <- LETTERS[1:10]
```

This generates a data matrix `X` with 10 covariates called `A`, `B`, ..., `J`, and a response `Y`. In truth, the covariates `A`, `B`, and `C` are associated with `Y`, and the rest are not.

We start the *globaltest* package

```
> library(globaltest)
```

2.1.2 Options

The *globaltest* package has a `gt.options` function, which can be used to set some global options of the package. We use this in this vignette to switch off the progress information, which is useful if the functions are used interactively, but does not combine well with `Sweave`, which was used to make this vignette. We also set the `max.print` option in *globaltest*, which abbreviates long Gene Ontology terms in Chapter 3.

```
> gt.options(trace = FALSE, max.print = 45)
```

2.1.3 The test

The main workhorse function of the *globaltest* package is the `gt` function, which performs the actual test. There are several alternative ways to call this function, depending on the user's preference to work with *formula* objects or matrices. We start with the *formula*-based way, because this is closest to the statistical theory. Matrix-based calls are detailed in Section 2.1.6.

In the data set of Section 2.1.1, if we are interested in testing for association between the group of variables `A`, `B` and `C` with the response `Y`, we can test the null hypothesis `Y ~ 1` that the response depends on none of the variables in the group, against the alternative hypothesis `Y ~ A + B + C` that `A`, `B` and `C` may have an influence on the response. We test this with

```
> gt(Y ~ 1, Y ~ A + B + C, data = X)

   p-value Statistic Expected Std.dev #Cov
1 2.29e-06      50.3     5.26    5.12    3
```

Unlike in `anova`, the order of the models matters in this call: the second argument must always be the alternative hypothesis.

The output lists the p-value of the test, the test statistic with its expected value and standard deviation under the null hypothesis. The `#Cov` column give the number of covariates in the alternative model that are not in the null model. In the linear model the test statistic is scaled in such a way that it takes values between 0 and 100. The test statistic can be interpreted as 100 times a weighted average (partial) correlation between the covariates of the alternative and the residuals of the response. In other models, the test statistic has a roughly similar scaling and interpretation.

2.1.4 Nuisance covariates

A similar syntax can be used to correct the test for nuisance covariates. To correct the test of the previous section for the possible confounding influence of the covariate `D`, we specify the null hypothesis `Y ~ D` versus the alternative `Y ~ A + B + C + D`. Note that the nuisance covariate occurs both in the null and alternative models.

```
> gt(Y ~ D, Y ~ A + B + C + D, data = X)

   p-value Statistic Expected Std.dev #Cov
1 8.47e-06      48.1     5.56    5.32    4
```

2.1.5 The *gt.object* object: extracting information

The `gt` function returns a *gt.object* object, which stores some useful information, for example the information to make diagnostic plots. Many methods have been defined for this object. One useful function is the `summary` method

```
> summary(gt(Y ~ A, Y ~ A + B + C, data = X))
```

```
"gt.object" object from package globaltest

Call:
gt(response = Y ~ A, alternative = Y ~ A + B + C, data = X)

Model: linear regression.
Degrees of freedom: 20 total; 2 null; 2 + 3 alternative.
Null distibution: asymptotic.

   p-value Statistic Expected Std.dev #Cov
1 0.000252      42.9     5.56    5.98    3
```

Other functions to extract useful information from a *gt.object*. For example,

```
> res <- gt(Y ~ A, Y ~ A + B + C, data = X)
> p.value(res)

[1] 0.0002522156

> z.score(res)

[1] 6.249677

> result(res)

       p-value Statistic Expected  Std.dev #Cov
1 0.0002522156  42.94048 5.555556 5.981898    3

> size(res)

#Cov
   3
```

The `z.score` function returns the test statistic standardized by its expectation and standard deviation under the null hypothesis; `result` returns a *data.frame* with the test result; `size` returns the number of alternative covariates.

2.1.6 Alternative function calls

The call to `gt` is quite flexible, and the null and alternative hypotheses can be specified using either *formula* objects or design matrices. We illustrate both types of calls, starting with the *formula*-based ones.

As the global test always tests nested models, there is no need to repeat the response and the null covariates when specifying the alternative model, so we may abbreviate the call of the previous section by specifying only those alternative covariates that do not already appear in the null model. Therefore,

```
> gt(Y ~ A, ~B + C, data = X)
```

also tests the null hypothesis `Y ~ A` versus the alternative `Y ~ A + B + C`.

If only a single model is specified, `gt` will test a null model with only an intercept against the specified model. So, to test the null hypothesis `Y ~ 1` against the alternative `Y ~ A + B + C`, we may write

```
> gt(Y ~ A + B + C, data = X)

   p-value Statistic Expected Std.dev #Cov
1 2.29e-06      50.3     5.26    5.12    3
```

The dot (`.`) argument for *formula* objects can often be useful. To test `Y ~ A` against the global alternative that all covariates are associated with `Y`, we can test

```
> gt(Y ~ A, ~., data = X)

  p-value Statistic Expected Std.dev #Cov
1 0.00454      16.0     5.56    2.97   10
```

Using the information from the column names in the *data* argument, the `~ .` argument is automatically expanded to `~ A + B + C + D + E + F + G + H + I + J`.

In some applications it is more natural to work with design matrices directly, rather than to specify them through a *formula*. To perform the test of `Y ~ 1` against `Y ~ .`, we may write

```
> gt(Y, X)

   p-value Statistic Expected Std.dev #Cov
1 7.34e-06      24.3     5.26    2.79   10
```

Similarly, the null hypothesis may be specified as a design matrix. The call

```
> designA <- cbind(1, X[, "A"])
> gt(Y, X, designA)

  p-value Statistic Expected Std.dev #Cov
1 0.00454      16.0     5.56    2.97   10
```

gives the same result as `gt(Y~A, ~., data = X)`, except for the `#Cov` output: the function cannot detect that some of the null covariates are also present in the alternative design matrix, only that the latter contains exactly correlated ones. Note that when specified in this way the null design matrix must be a complete design matrix, i.e. with any intercept term included in the matrix.

2.1.7 Models

The `gt` function can work with the following models: linear regression, logistic regression and multinomial logistic regression, poisson regression and the Cox proportional hazards model. The model to be used can be specified by the *model* argument.

```
> P <- rpois(20, lambda = 2)
> gt(P ~ A, ~., data = X, model = "Poisson")

  p-value Statistic Expected Std.dev #Cov
1    0.72      4.23     6.07    2.99   10

> gt(P ~ A, ~., data = X, model = "linear")

  p-value Statistic Expected Std.dev #Cov
1   0.814      3.09     5.56    2.97   10
```

If the null model has no covariates (i.e. `~0` or `~1`), the logistic and Poisson model results are identical to the linear model results.

If missing, the function will try to determine the model from the input. If the response is a *factor* with two levels or a *logical*, it uses a logistic model; if a factor with more than two levels, a multinomial logistic model; if the response is a *Surv* object, it uses a Cox model (for examples, see Section 3.5). In all other cases the default is linear regression.

Use `summary` to check which model was used.

2.1.8 Null distribution: asymptotic or permutations

By default the global test uses an analytic null distribution to calculate the p-values of the test. This analytic distribution is exact in case of the linear model with normally distributed errors, and asymptotic in all other models. The distribution that is used is described in Goeman et al. (2009) for linear and generalized linear models, and in Goeman et al. (2005) for the Cox proportional hazards model. The assumption underlying the asymptotic distribution is that the sample size is (much) larger than the number of covariates of the null hypothesis; the dimensionality of the alternative is not an issue.

For the linear, logistic and poisson models, the reported p-values are numerically reliable up to at least two decimal places down to values of around 10^{-12}. Reported lower p-values are less reliable (although they can be trusted to be below 10^{-12}).

In situations in which the assumptions underlying the asymptotics are questionable, or in which an exact alpha level of the test is necessary, it is possible to calculate the p-value using permutations instead. Because permutations require an exchangeable null hypothesis, such a permutation p-value is only available for the linear model and for the exchangeable null hypotheses `~1` and `~0` in other models.

To calculate permutation p-values, specify the number of permutations with the *permutations* argument. The default, `permutations = 0`, selects the asymptotic distribution. If the number of permutations specified in *permutations* is larger than the total number of possible permutations, all possible permutations are used; otherwise the function draws permutations at random. Use `summary` to see which variant was actually used.

Compare

```
> gt(Y, X)
```

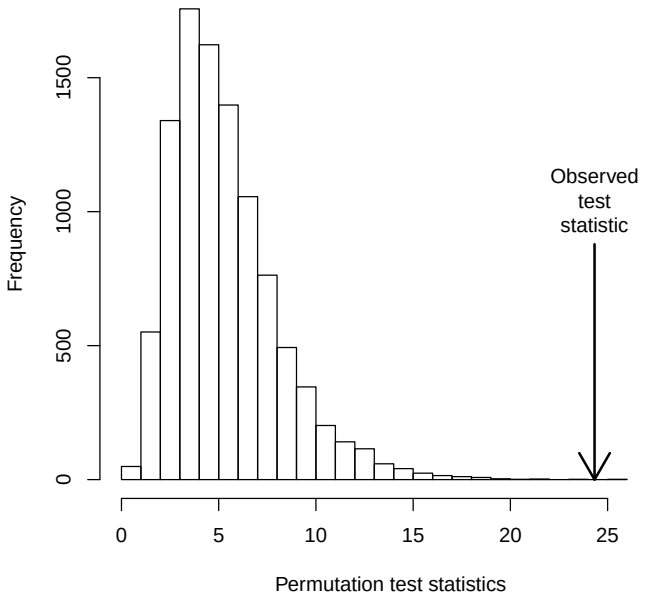
```
   p-value Statistic Expected Std.dev #Cov
1 7.34e-06      24.3     5.26    2.79   10
```

```
> gt(Y, X, permutations = 10000)
```

```
  p-value Statistic Expected Std.dev #Cov
1       0      24.3     5.27    2.69   10
```

The distribution of the permuted test statistic can be visualized using the `hist` function.

```
> hist(gt(Y, X, permutations = 10000))
```



2.1.9 Intercept terms

If *null* is given as a *formula* object, intercept terms are automatically included in the model unless this term is explicitly removed with `~0+...` or `~...-1`, as is usual in *formula* objects. This automatic addition of an intercept does not happen if *null* is specified as a design matrix. Therefore, the calls

```
> A <- X[, "A"]
> gt(Y, X, A)
```

```
  p-value Statistic Expected Std.dev #Cov
1 0.00531      15.2     5.26    2.87   10
```

```
> gt(Y, X, ~A)

  p-value Statistic Expected Std.dev #Cov
1 0.00454      16.0     5.56    2.97   10
```

test different null hypotheses: `Y ~ 1 + A` and `Y ~ 0 + A`, respectively.

In contrast, in the alternative model the intercept term is always suppressed, even if *alternative* is a *formula* and an intercept is not present in the null model. If a user wants to include an intercept term in the alternative model but not in the null model, he must explicitly construct an intercept variable. The reason for this is that the test result is not invariant to the scaling of variables in the alternative, and therefore also not invariant to relative scaling of the intercept to the other variables. The user must therefore choose and construct an appropriately scaled intercept. The call

```
> gt(Y ~ 0 + A, ~B + C, data = X)

   p-value Statistic Expected Std.dev #Cov
1 0.000140      43.8     5.26    5.72    2
```

suppresses the intercept both in null and alternative hypotheses. To include an intercept in the alternative, we must say something like

```
> IC <- rep(1, 20)
> gt(Y ~ 0 + A, ~IC + B + C, data = X)

   p-value Statistic Expected Std.dev #Cov
1 0.000228      32.9     5.26    4.59    3
```

Note that setting `IC <- rep(2, 20)` gives a different result.

2.1.10 Covariates of class *factor*

Another consequence of the fact that the global test is not invariant to the parametrization of the alternative model is that one must carefully consider the choice of contrasts for *factor* covariates. We distinguish nominal (unordered) factors and ordinal (ordered) factors.

The usual coding of nominal factors with a reference category and dummy variables that describe the difference between each category and the reference is usually not appropriate for global test, as this parametrization (and therefore the test result) depends on the choice of the reference category, which is often arbitrary. More appropriate is to do a symmetric parametrization with a dummy for each category. This works even if multiple factors are considered, because the global test is not adversely affected by overparametrization. If `gt` was called with the argument *x* set to `TRUE`, we can use `model.matrix` on the *gt.object* to check the design matrix.

```
> YY <- rnorm(6)
> FF <- factor(rep(letters[1:2], 3))
> GG <- factor(rep(letters[3:5], 2))
> model.matrix(gt(YY ~ FF + GG, x = TRUE))$alternative

  FFa FFb GGc GGd GGe
1   1   0   1   0   0
2   0   1   0   1   0
3   1   0   0   0   1
4   0   1   1   0   0
5   1   0   0   1   0
6   0   1   0   0   1
```

This choice of contrasts guarantees that the test result does not depend on the order of the levels of any factors.

For ordered factors it is often appropriate to make contrasts between successive categories. For example if a factor has three ordered categories `a`, `b`, and `c`, one contrast `a<b` codes the difference between categories `a` on the one hand and `b`, and `c` on the other, whereas `b<c` codes the difference between categories `a` and `b` on the one hand and `c` on the other.

```
> GG <- ordered(GG)
> model.matrix(gt(YY ~ GG, x = TRUE))$alternative

  GGc<d GGd<e
1    -1     0
2     0     0
3     0     1
4    -1     0
5     0     0
6     0     1
```

This effectively takes the middle category to be the reference category, and assumes that the effects of categories further apart are more diverse than effects of categories closer to each other. In case of an even number of categories a latent category is created between the two middlemost categories, leading to a slightly more intricate contrasts matrix.

2.1.11 Directing the test: weights

The global test assigns relative weights to each covariate in the alternative which determine the contribution of each covariate to the test result. The default weighting, which follows from the theory of the test (Goeman et al., 2006), is proportional to the residual variance of each of the covariates, after orthogonalizing them with respect to the null covariates. The weights that `gt` uses internally can be retrieved with the `weights` function.

```
> res <- gt(Y, X)
> weights(res)

        A         B         C         D         E         F         G         H
0.6462082 1.0000000 0.8522877 0.4298123 0.3435935 0.2312562 0.7261093 0.4916427
        I         J
0.4260604 0.6629415
```

Only the ratios between weights are relevant. The weights that are returned are scaled so that the maximum weight is 1.

In some applications the default weighting is not appropriate, for example if the covariates are all measured in different units and the relative scaling of the units is arbitrary. In that case it is better to standardize all covariates to unit standard deviation before performing the test. This can be done using the *standardize* argument.

```
> res <- gt(Y, X, standardize = TRUE)
> weights(res)

A B C D E F G H I J
1 1 1 1 1 1 1 1 1 1
```

Alternatively, the function can work with user-specified weights, given in the *weights* argument. These weights are multiplied with the default weights, unless the *standardize* argument is set to `TRUE`. The following two calls give the same test result.

```
> gt(Y, X[, c("A", "A", "B")], weights = c(0.5, 0.5, 1))
> gt(Y, X[, c("A", "B")])
```

2.1.12 Directing the test: directional

The power of the global test does not depend on the sign of the true regression coefficients. However, in some applications the regression coefficients of different covariates are a priori expected to have the same sign. Using the *directional* argument The test can be directed to be more powerful against the alternative that the regression coefficients under the alternative all have the same sign.

```
> gt(Y, X, directional = TRUE)

  p-value Statistic Expected Std.dev #Cov
1 0.00156      31.3     5.26       5   10
```

In the hierarchical model formulation of the test, this is achieved by making the random regression coefficients a priori positively correlated. The default, `directional = TRUE`, corresponds to an a priori correlation between regression coefficients of $\sqrt{1/2}$. If desired, the *directional* argument can be set to a value other than `TRUE`. Setting *directional* to a value of d corresponds to an a priori correlation of $\sqrt{d/(1+d)}$.

If some covariates are a priori expected to have regression coefficients with opposite signs, the corresponding covariates can be given negative weights.

2.1.13 Offset terms and testing values other than zero

By default, the global test tests the null hypothesis that all regression coefficients of the covariates of the alternative hypothesis are all zero. It is also possible to test the null hypothesis that these covariates have a different value than zero, specified by the user. This can be done using the *test.value* argument.

```
> gt(Y ~ A + B + C, data = X, test.value = c(0.2, 0.2, 0.2))

  p-value Statistic Expected Std.dev #Cov
1   0.156      9.33     5.26    5.12    3
```

The *test.value* argument is always applied to the original alternative design matrix, i.e. before any standardization or weighting.

Specifying *test.value* in this way is equivalent to adding an `offset` term to the null hypothesis of $X\mathbf{v}$, where X is the design matrix of the alternative hypothesis and $\mathbf{v}$ is the specified *test.value*.

```
> os <- X[, 1:3] %*% c(0.2, 0.2, 0.2)
> gt(Y ~ offset(os), ~A + B + C, data = X)

  p-value Statistic Expected Std.dev #Cov
1   0.156      9.33     5.26    5.12    3
```

Offset terms are not implemented for the multinomial logistic model.

2.2 Diagnostic plots

Aside from the permutations histogram already mentioned in Section 2.1.8, there are two main diagnostic plots that can help users to interpret a test result. Both plots are based on a decomposition of the test result into component test statistics that only use part of the information that the full test uses.

2.2.1 The `covariates` plot

As shown in Goeman et al. (2004), the global test statistic on a collection of alternative covariates can be seen as a weighted average of the global test statistics for each individual alternative covariate.

```
> gt(Y ~ A + B, data = X)

   p-value Statistic Expected Std.dev #Cov
1 2.05e-07      58.4     5.26    5.58    2

> gt(Y ~ A, data = X)

  p-value Statistic Expected Std.dev #Cov
1   0.002        42     5.26    7.24    1
```

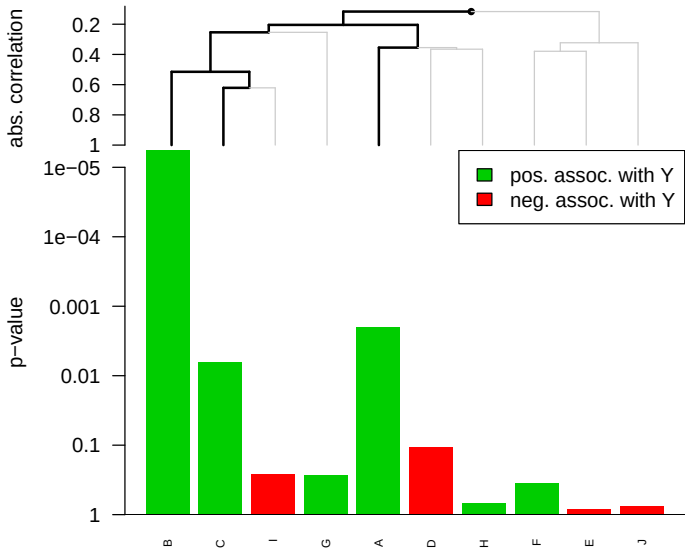
```
> gt(Y ~ B, data = X)

   p-value Statistic Expected Std.dev #Cov
1 5.72e-06        69     5.26    7.24    1
```

The test statistic of the test against `~A+B` is between the test statistics against the alternatives `~A` and `~B`, even though the cumulative evidence of `A` and `B` may make the p-value of the combined test smaller than that of each individual one. This is because the global test statistic for an alternative hypothesis is always a weighted average of the test statistics for tests of the component single covariate alternatives. The `covariates` plot is based on this decomposition of the test statistic into the contributions made by each of the covariates in the alternative hypothesis.

The contribution of each such covariate is itself a test. It can be useful to make a plot of these test results to find those covariates or groups of covariates that contribute most to a significant test result.

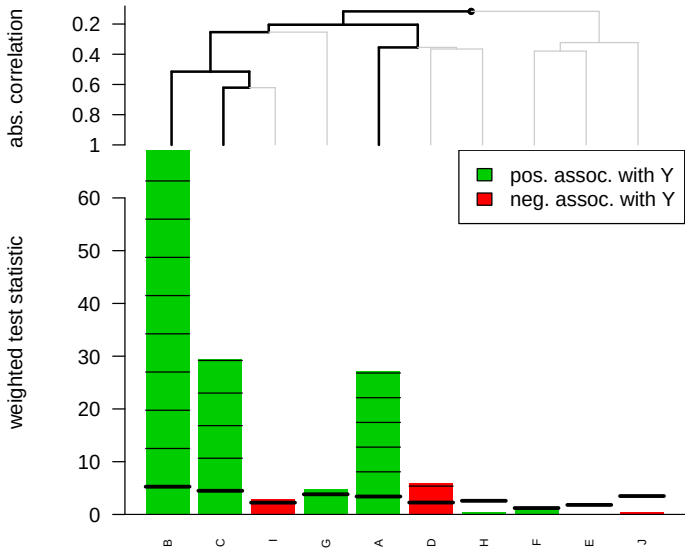
```
> covariates(gt(Y, X))
```



The `covariates` plot by default plots the p-values of the tests of individual component covariates of the alternative. Other characteristic values of the component tests may be plotted using the *what* argument: specifying `what = "z"` plots standardized test statistics (compare the `z.score` method for *gt.object* objects); specifying

`what = "s"` gives the unstandardized test statistics and `what = "w"` give the unstandardized test statistics weighted for the relative weights of the covariates in the test (compare the `weights` method for *gt.object* objects). If (weighted or unweighted) test statistics are plotted, bars and stripes appear to signify mean and standard deviation of the bars under the null hypothesis.

```
> covariates(gt(Y, X), what = "w")
```



The plotted covariates are ordered in a hierarchical clustering graph. The distance measure used for the graph is absolute correlation distance if the *directional* argument of `gt` was `FALSE` (the default), or correlation distance otherwise. (Absolute) correlation distance is appropriate here because the test results for the individual covariates can be expected to be similar if the covariates are strongly correlated, and because the sign of the correlation matters only if a directional test was used. The default clustering method is average linkage. This can be changed if desired, using the *cluster* argument. Clustering can also be turned off by setting `cluster = FALSE`.

The hierarchical clustering graph induces a collection of subsets of the tested covariates between the full set that is the top of the clustering graph and the single covariates that are the leaves. There are $2k - 1$ such sets for a graph with k leave nodes, including top and leaves. It is possible to do a multiple testing procedure on all $2k-1$ sets, controlling the family-wise error rate while taking the structure of the graph into account. The `covariates` function performs such a procedure, called the *inheritance*

procedure, which is an adaptation of the method of Meinshausen (2008): see Section 2.3.4. By coloring the part of the clustering graph that has a significant multiplicity-corrected p-value in black, the user can get an impression what covariates and clusters of covariates are most clearly associated with the response variable. The significance threshold at which a multiplicity-corrected p-value is called significant can be adjusted with the *alpha* argument (default 0.05). In some situations the significant branches do not reach all the way to the leaf nodes. The interpretation of this is that the multiple testing procedure can infer with confidence that at least one of the covariates below the last significant branch is associated with the response, but it cannot pinpoint with enough confidence which one(s).

The result of the covariates function can be stored to access the information in the graph. The `covariates` function returns a *gt.object* containing all tests on all subsets induced by the clustering graph, with their familywise error adjusted p-values.

```
> res <- covariates(gt(Y, X))
> res[1:10]
```

	alias	inheritance	p-value	Statistic	Expected	Std.dev	#Cov
O		7.34e-06	7.34e-06	24.33	5.26	2.79	10
O[1		7.34e-06	4.94e-06	30.56	5.26	3.44	7
O[1[1		1.00e-04	5.19e-05	35.37	5.26	4.46	4
O[1[1[1		1.53e-04	6.02e-05	44.50	5.26	5.37	3
O[1[1[1[1:B	B	1.53e-04	5.72e-06	69.04	5.26	7.24	1
O[1[1[1[2		3.46e-02	1.36e-02	25.31	5.26	6.11	2
O[1[1[1[2[1:C	C	3.46e-02	6.47e-03	34.49	5.26	7.24	1
O[1[1[1[2[2:I	I	6.68e-01	2.62e-01	6.93	5.26	7.24	1
O[1[1[2:G	G	6.68e-01	2.70e-01	6.70	5.26	7.24	1
O[1[2		2.82e-02	7.62e-03	21.36	5.26	4.56	3

The names of the subsets should be read as follows. Öřefers to the origin or root, and each "[1" refers to a first (or left) branch, whereas each "[2" refers to a second (or right) branch. Leaf nodes are also referred to by name. To get the leaf nodes of the subgraph that is significant after multiple testing correction, use the `leafNodes` function

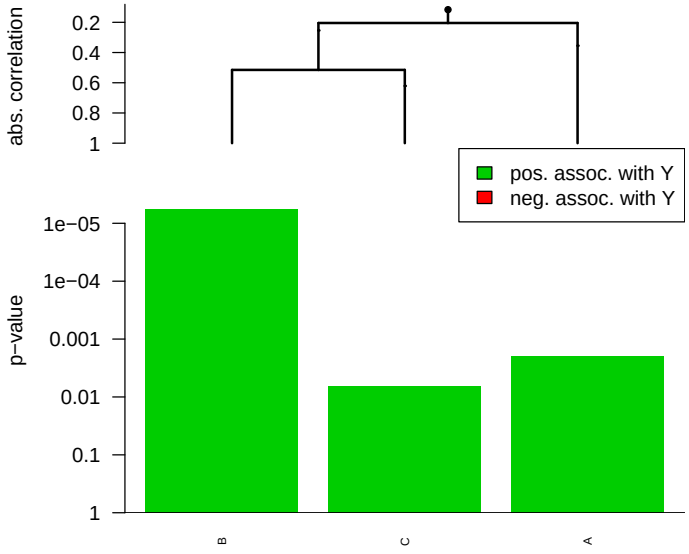
```
> leafNodes(res, alpha = 0.1)
```

	alias	inheritance	p-value	Statistic	Expected	Std.dev	#Cov
O[1[1[1[1:B	B	0.000153	5.72e-06	69.0	5.26	7.24	1
O[1[1[1[2[1:C	C	0.034561	6.47e-03	34.5	5.26	7.24	1
O[1[2[1:A	A	0.028237	2.00e-03	42.0	5.26	7.24	1

The function tries to sort the bars in such a way that the most significant covariates appear on the left. This sorting is, of course, constrained by the dendrogram if present. Setting the *sort* argument to `FALSE` to keep the bars in the original order as much as possible under the same constraints.

An additional option *zoom* is available that "zooms in" on the significant branches by discarding the non-significant ones. If the whole graph is non-significant *zoom* has no effect.

```
> covariates(gt(Y, X), zoom = TRUE)
```



The default colors, legend and labels in the plot can be adjusted with the *colors*, *legend* and *alias* arguments.

The covariates returns the test results for all tests it performs, invisibly, as a *gt.object*. The `leafNodes` function can be used to extract useful information from this object. Using `leafNodes` with the same value of alpha that was used in the covariates function, extracts the test results for the leaves of the significant subgraph. Using `alpha = 1` extracts the test results for leaves of the full graph, i.e. for the individual covariates.

By default, the `covariates` function can only make a plot for a single test result, even if the *gt.object* contains multiple test results (see Section 2.3.1). However, by providing a filename in the *pdf* argument of the `covariates` function it is possible to make multiple plots, writing them to a pdf file as separate pages.

Those who like a more machine-learning oriented terminology can use the `features` function, which is identical to `covariates` in all respects.

2.2.2 The `subjects` plot

Alternatively, it is possible to visualize the influence of the subjects, rather than of the covariates, on the test result. This can be useful in order to look for subjects that have

an overly large influence on the test result, of to find subjects that deviate from the main pattern.

Visualizing the test result in terms of the contributions of the subjects can be done using a different decomposition of the test result. In the linear model the test statistic Q can be viewed as a weighted sum of the quantities

$$Q_i = \text{sign}(Y_i - \mu_i) \sum_{j=1}^{n} \sum_{k=1}^{p} X_{ik} X_{jk} (Y_j - \mu_j),$$

where Y_i is the response variable of subject i, μ_i that person's expected response under the null hypothesis, and X the design matrix of the alternative. We subtract $\hat{\text{E}}(Q_i) = \text{sign}(Y_i - \mu_i) \sum_{k=1}^{p} X_{ik} X_{ik} (Y_i - \mu_i)$ as a crude estimate of the expectation of Q_i. An estimate of the variance of Q_i is $\text{Var}\{Q_i - \hat{\text{E}}(Q_i)\} = \sigma^2 \sum_{j=1}^{n} \sum_{k=1}^{p} X_{ik}^2 X_{jk}^2$. The quantities are asymptotically normally distributed. A similar decomposition can be made for the test statistic in other models than the linear one.

The resulting quantity $Q_i - \hat{\text{E}}(Q_i)$ can be interpreted as the contribution of the i-th subject to the test statistic in the sense that it is proportional to the difference between the test statistic for the full sample and the test statistic of a reduced sample in which subject i has been removed. It can also be interpreted as an alternative test statistic for the same null hypothesis as the global test, but one which uses only part of the information that the full global test uses.

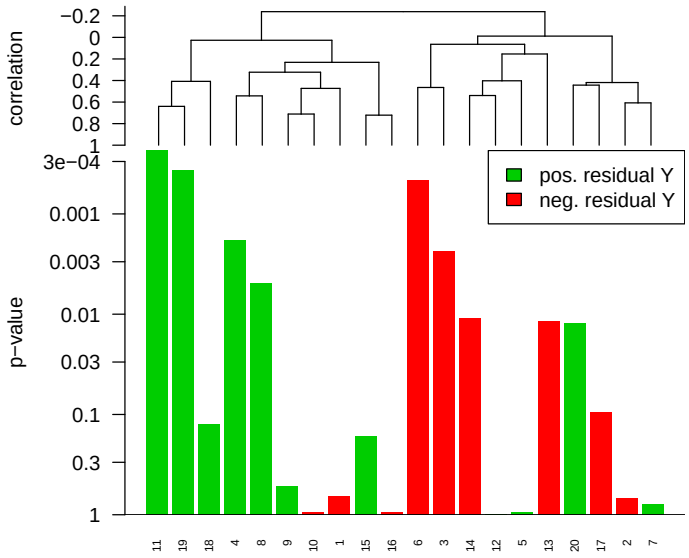
The contribution $Q_i - \hat{\text{E}}(Q_i)$ of individual i takes a large value if other subjects who are similar to subject i in terms of their covariates X (measured in correlation distance) also tend to be similar in terms of their residual $Y_j - \mu_j$ (i.e. has the same sign). This contribution $Q_i - \hat{\text{E}}(Q_i)$ can, therefore, be viewed as a partial global test statistic that rejects if individuals that are similar to individual i in terms of their alternative covariates tend to deviate from the null model in the same direction as individual i with their response variable.

The `subjects` function plots the p-values of these partial test statistics. As in the `covariates` function, other values may be plotted using the *what* argument. Specifying `what = "z"` plots test statistics standardized by their expectation and standard deviation; specifying `what = "s"` gives the unstandardized test statistics Q_i and `what = "w"` give the unstandardized test statistics weighted for the relative weights of the subjects in the test (proportional to $|Y_i|$). If weighted or unweighted standardized test statistics are plotted, bars and stripes appear to signify mean and standard deviation of the bars under the null hypothesis.

An additional argument *mirror* (default: `TRUE`) can be used to plot the unsigned version $\bar{Q}_i = \sum_{j=1}^{n} \sum_{k=1}^{p} X_{ik} X_{jk} (Y_j - \mu_j)$ (no effect if `what = "p"`). Combined with `what = "s"`, this gives the first partial least squares component of the data, which can be interpreted as a first order approximation of the estimated linear predictor under the alternative. In the resulting plot, large positive values correspond to subjects that have a much higher predicted value under the alternative hypotheses than under the null, whereas large negative values correspond to subjects with a much lower expected value under the alternative than under the null.

As in the `covariates` plot, the subjects in the `subjects` plot are ordered in a hierarchical clustering graph. The distance measure used for the clustering graph

```
> subjects(gt(Y, X))
```



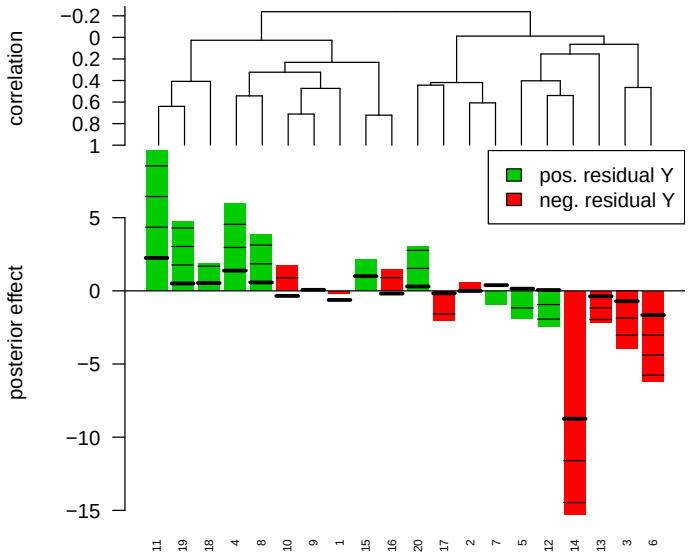
is correlation distance. Correlation distance is appropriate because the test results for subjects can be expected to be similar if their measurements are close in terms of correlation distance. The default clustering method is average linkage. This can be changed if desired, using the *cluster* argument. Clustering can also be turned off by setting `cluster = FALSE`. Unlike in the `covariates` plot, no multiple testing is done on the clustering graph.

The function tries to sort the bars in such a way that the most significant partial tests appear on the left. This sorting is, of course, constrained by the dendrogram if present. Setting the *sort* argument to `FALSE` to keep the bars in the original order as much as possible under the same constraints.

The default colors, legend and labels in the plot can be adjusted with the *colors*, *legend* and *alias* arguments.

By default, the `subjects` function can only make a plot for a single test result, even if the *gt.object* contains multiple test results (see Section 2.3.1). However, by providing a filename in the *pdf* argument of the `subjects` function it is possible to make multiple plots, writing them to a pdf file as separate pages.

```
> subjects(gt(Y, X), what = "s", mirror = FALSE)
```



2.3 Doing many tests: multiple testing

In high-dimensional data, when the dimensionality of the design matrix of the alternative is very high, it is often interesting to study subsets of the covariates, or to compare alternative weighting options. The *globaltest* package facilitates this by making it possible to perform tests for many alternatives at once, and to perform various algorithms for multiple testing correction.

2.3.1 Many subsets or many weights

To test one or many subsets covariates of the alternative design matrix, use the *subsets* argument. If a single subset is to be tested, the *subsets* argument can be presented as a vector of covariate names or of covariate indices in the alternative design matrix.

```
> set <- LETTERS[1:3]
> gt(Y, X, subsets = set)

   p-value Statistic Expected Std.dev #Cov
1 2.29e-06      50.3     5.26    5.12    3
```

To test many subsets, *subsets* can be a (named) list of such vectors.

```
> sets <- list(one = LETTERS[1:3], two = LETTERS[4:6])
> gt(Y, X, subsets = sets)

     p-value Statistic Expected Std.dev #Cov
one 2.29e-06     50.26     5.26    5.12    3
two 2.63e-01      7.09     5.26    4.23    3
```

Duplicate identifiers in the subset vectors are not removed, but lead to increased weight for the duplicated covariates in the resulting test, except if the `trim` option was set to `TRUE` (see Section 3.2.3).

To retrieve the subsets from a *gt.object*, use the `subsets` method.

```
> res <- gt(Y, X, subsets = sets)
> subsets(res)

$one
[1] "A" "B" "C"

$two
[1] "D" "E" "F"
```

Weighting was already discussed in Section 2.1.11. To test many different weights simultaneously, the *weights* argument can also be given as a (named) list, similar to the subsets argument.

```
> wts <- list(up = 1:10, down = 10:1)
> gt(Y, X, weights = wts)

      p-value Statistic Expected Std.dev #Cov
up   1.83e-02      11.9     5.26    2.73   10
down 1.51e-06      35.0     5.26    3.50   10
```

Weights can also be used as an alternative way of specifying subsets, by giving weight 1 to included covariates and 0 to others.

Weights and subsets can also be combined. Either specify a single weights vector for many subsets

```
> gt(Y, X, subsets = sets, weights = 1:10)

     p-value Statistic Expected Std.dev #Cov
one 2.02e-05     48.70     5.26    5.47    3
two 3.12e-01      6.39     5.26    4.17    3
```

or specify a separate weights vector for each subset. In the latter case case each weights vector may be either a vector of the same length as the number of covariates in the alternative design matrix, or, alternatively, be equal in length to corresponding subset.

```
> gt(Y, X, subsets = sets, weights = wts)
```

```
    alias  p-value Statistic Expected Std.dev #Cov
one    up 2.02e-05     48.70     5.26    5.47    3
two  down 2.30e-01      7.63     5.26    4.36    3
```

```
> gt(Y, X, subsets = sets, weights = list(1:3, 7:5))
```

```
     p-value Statistic Expected Std.dev #Cov
one 2.02e-05     48.70     5.26    5.47    3
two 2.30e-01      7.63     5.26    4.36    3
```

Note that in case of a name conflict between the *subsets* and *weights* arguments, the names of the *weights* argument are returned under "alias". In general, the alias is meant to store additional information on each test performed. Unlike the name, the alias does not have to be unique. An alias for the test result may be provided with the *alias* argument, or added or changed later using the `alias` method.

```
> res <- gt(Y, X, weights = wts, alias = c("one", "two"))
> alias(res)
```

```
[1] "one" "two"
```

```
> alias(res) <- c("ONE", "TWO")
```

To take a subset of the test results, a `gt.object` can be subsetted using `[` or `[[` as with other `R` objects. There is no distinction between `[` or `[[`. A `gt.object` can be sorted to increasing p-values with the `sort` command. In case of equal p-values, which may happen e.g. when doing permutation testing, the tests with the same p-values are sorted to decreasing z-scores.

```
> res[1]
```

```
  alias p-value Statistic Expected Std.dev #Cov
1   ONE  0.0183      11.9     5.26    2.73   10
```

```
> sort(res)
```

```
  alias  p-value Statistic Expected Std.dev #Cov
2   TWO 1.51e-06      35.0     5.26    3.50   10
1   ONE 1.83e-02      11.9     5.26    2.73   10
```

2.3.2 Unstructured multiple testing procedures

When doing many tests, it is important to correct for multiple testing. The *globaltest* package offers different methods for correcting for multiple testing. For unstructured tests in which the tests are simply considered as an exchangeable list with no inherent structure. These methods are described in the help file of the `p.adjust` function (*stats* package). The three most important ones are

Holm The procedure of Holm (1979) for control of the family-wise error rate

BH The procedure of Benjamini and Hochberg (1995) for control of the false discovery rate

BY The procedure of Benjamini and Yekutieli (2001) for control of the false discovery rate

The procedures of Holm and Benjamini and Yekutieli (2001) are valid for any dependency structure between the null hypotheses, but the procedure of Benjamini and Hochberg (1995) is only valid for independent or positively correlated test statistics (see Benjamini and Yekutieli, 2001, for details).

Multiplicity-corrected p-values can be calculated with the `p.adjust` function. The default procedure is Holm's procedure.

```
> p.adjust(res)
```

```
     alias     holm  p-value Statistic Expected Std.dev #Cov
up     ONE 1.83e-02 1.83e-02      11.9     5.26    2.73   10
down   TWO 3.03e-06 1.51e-06      35.0     5.26    3.50   10
```

```
> p.adjust(res, "BH")
```

```
     alias       BH  p-value Statistic Expected Std.dev #Cov
up     ONE 1.83e-02 1.83e-02      11.9     5.26    2.73   10
down   TWO 3.03e-06 1.51e-06      35.0     5.26    3.50   10
```

```
> p.adjust(res, "BY")
```

```
     alias       BY  p-value Statistic Expected Std.dev #Cov
up     ONE 2.74e-02 1.83e-02      11.9     5.26    2.73   10
down   TWO 4.54e-06 1.51e-06      35.0     5.26    3.50   10
```

2.3.3 Graph-structured hypotheses 1: the focus level method

Sometimes the sets of covariates that are to be tested are structured in such a way that some sets are subsets of other sets. Such a structure can be exploited to gain improved power in a multiple testing procedure. The *globaltest* package offers two procedures that make use of the structure of the sets when controlling the familywise error rate. These procedures are the focus level procedure of Goeman and Mansmann (2008), and the inheritance procedure, a variant of the procedure of Meinshausen (2008). We treat both of these methods in turn.

Sets of covariates can be viewed as nodes in a graph, with subset relationships form the directed edges. Viewed in this way, any collection of covariates forms a directed acyclic graph. The inheritance procedure is restricted to tree-structured graphs. The focus level is not so restricted, and can work with any directed acyclic graph.

To illustrate the focus level method, let's make some covariate sets of interest.

```
> level1 <- as.list(LETTERS[1:10])
> names(level1) <- letters[1:10]
```

```
> level2 <- list(abc = LETTERS[1:3], cde = LETTERS[3:5], fgh = LETTERS[6:8],
      hij = LETTERS[8:10])
> level3 <- list(all = LETTERS[1:10])
> dag <- c(level1, level2, level3)
```

This gives one top node, 10 leaf nodes and 4 intermediate nodes. The structure is a directed acyclic graph because leaf nodes "C" and "H" both have more than one parent.

The focus level method requires the choice of a *focus level*. This is the level in the graph at which the procedure starts testing. If significant nodes are found at this level, the procedure will fan out to find significant ancestors and offspring of that significant node. A focus level can be specified as a character vector of node identifiers, or it can be generated in an automated way using the `findFocus` function.

```
> fl <- names(level2)
> fl <- findFocus(dag, maxsize = 8)
```

The `findFocus` function chooses the focus level in such a way that each focus level node has at most *maxsize* non-redundant offspring nodes, where a redundant node is a node that can be constructed as a union of other nodes. An optional argument *atoms* (default: `TRUE`) first decomposes all nodes into *atoms*: small sets from which all offspring sets can be reconstructed as unions of atoms. Making use of these atoms often reduces computation time considerably, although it may, in theory, result in some loss of power.

To apply the focus level method, first create a *gt.object* that contains all the covariates under the alternative, e.g. the *gt.object* that uses the full alternative design matrix.

```
> res <- gt(Y, X)
> res <- focusLevel(res, sets = dag, focus = fl)
> sort(res)
```

```
    focuslevel  p-value Statistic Expected Std.dev #Cov
abc   9.17e-06 2.29e-06    50.260     5.26    5.12    3
all   9.17e-06 7.34e-06    24.327     5.26    2.79   10
b     6.69e-05 5.72e-06    69.036     5.26    7.24    1
a     8.01e-03 2.00e-03    41.998     5.26    7.24    1
c     2.59e-02 6.47e-03    34.494     5.26    7.24    1
cde   2.59e-02 9.15e-03    21.776     5.26    4.77    3
d     7.13e-01 1.07e-01    13.754     5.26    7.24    1
i     1.00e+00 2.62e-01     6.931     5.26    7.24    1
g     1.00e+00 2.70e-01     6.704     5.26    7.24    1
f     1.00e+00 3.51e-01     4.856     5.26    7.24    1
fgh   1.00e+00 4.70e-01     4.438     5.26    4.47    3
h     1.00e+00 6.92e-01     0.895     5.26    7.24    1
hij   1.00e+00 7.41e-01     2.387     5.26    4.05    3
j     1.00e+00 7.51e-01     0.573     5.26    7.24    1
e     1.00e+00 8.30e-01     0.263     5.26    7.24    1
```

As the `p.adjust` function, the `focusLevel` function reports familywise error rate adjusted p-values.

It is a property of both the inheritance and the focus level method, that the adjusted p-value of a node can never be smaller than a p-value of an ancestor node. The significant graph at a certain significance level is therefore always a coherent graph, which always contains all ancestor nodes of any rejected node. Such a graph can be succinctly summarized by reporting only its leaf nodes. This can be done using the `leafNodes` function.

```
> leafNodes(res)
```

```
  focuslevel  p-value Statistic Expected Std.dev #Cov
a   8.01e-03 2.00e-03      42.0     5.26    7.24    1
b   6.69e-05 5.72e-06      69.0     5.26    7.24    1
c   2.59e-02 6.47e-03      34.5     5.26    7.24    1
```

The *alpha* argument of the `leafNodes` function can be used to specify the rejection threshold for the familywise error of the significant graph.

To visualize the test result as a graph, use the `draw`. By default, this function draws the graph with the significant nodes in black and the non-significant ones in gray. The *alpha* argument can be used to change the significance threshold. Alternatively, it is possible to draw only the significant subgraph, setting the *sign.only* argument to `TRUE`. The *names* argument (default `FALSE`) forces the use of names in the nodes. This can quickly become unreadable even for small graphs if the names for the nodes are long. By default, therefore, `draw` numbers the nodes, returning a legend to interpret the numbers.

```
> legend <- draw(res)
```

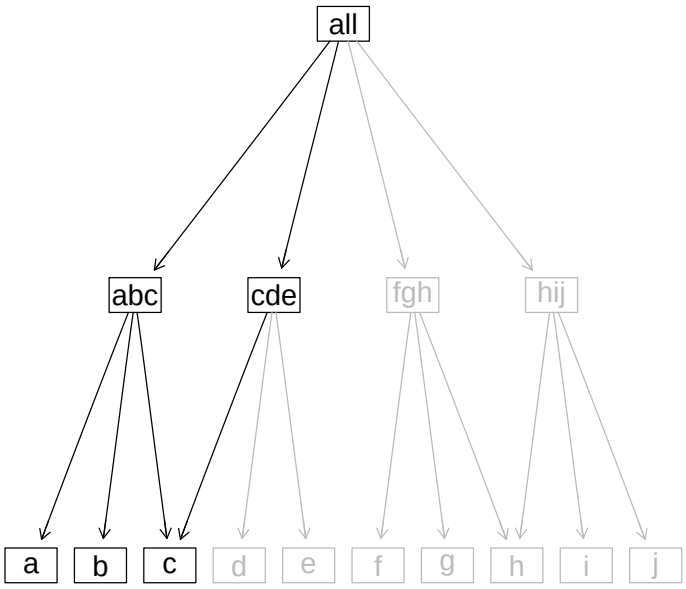
The *interactive* argument can be used to make the plot interactive. In an interactive plot, click on a node to see the node label. Exit the interactive plot by pressing escape.

2.3.4 Graph-structured hypotheses 2: the inheritance method

An alternative method for multiple testing in graph-structured hypotheses is the inheritance method. This procedure is based on the work of Meinshausen (2008). `inheritance` reports familywise error rate adjusted p-values, as `p.adjust` and `focusLevel` functions do. Compared with the focus level method, the inheritance procedure is less computationally intensive, and does not require the definition of any (focus) level. However, it requires that the graph has a tree structure, rather than the more general directed acyclic graph structure that the focus level works with.

To illustrate the inheritance method, we make use of the example data. However, we can not make uso of the `dag` object created in Section 2.3.3 since it does not have a tree structure. For example, `c` in `dag` is a descendant of both `abc` and `cde`. We modify the commands of the previous section to make sure that each element of `dag` has (at maximum) one parent; this guarantees that it is a tree-structured graph.

```
> draw(res, names = TRUE)
```




```
> level1 <- as.list(LETTERS[1:10])
> names(level1) <- letters[1:10]
> level2 <- list(ab = LETTERS[1:2], cde = LETTERS[3:5], fg = LETTERS[6:7],
      hij = LETTERS[8:10])
> level3 <- list(all = LETTERS[1:10])
> tree <- c(level1, level2, level3)
```

Now we can apply the `inheritance` method. The syntax of the function is very similar to the `focusLevel` function.

```
> res <- gt(Y, X)
> resI <- inheritance(res, tree)
> resI

    inheritance  p-value Statistic Expected Std.dev #Cov
a      7.06e-03 2.00e-03    41.998     5.26    7.24    1
b      2.02e-05 5.72e-06    69.036     5.26    7.24    1
c      2.49e-02 6.47e-03    34.494     5.26    7.24    1
d      4.95e-01 1.07e-01    13.754     5.26    7.24    1
e      1.00e+00 8.30e-01     0.263     5.26    7.24    1
f      1.00e+00 3.51e-01     4.856     5.26    7.24    1
```

```
g      1.00e+00 2.70e-01     6.704     5.26    7.24    1
h      1.00e+00 6.92e-01     0.895     5.26    7.24    1
i      1.00e+00 2.62e-01     6.931     5.26    7.24    1
j      1.00e+00 7.51e-01     0.573     5.26    7.24    1
ab     7.34e-06 2.05e-07    58.422     5.26    5.58    2
cde    2.34e-02 9.15e-03    21.776     5.26    4.77    3
fg     1.00e+00 2.93e-01     6.258     5.26    5.90    2
hij    1.00e+00 7.41e-01     2.387     5.26    4.05    3
all    7.34e-06 7.34e-06    24.327     5.26    2.79   10
```

The inheritance procedure has two variants: one with and one without the *Shaffer* variant (Meinshausen, 2008). Setting the argument `Shaffer = TRUE` allows uniform improvement of the power of the procedure, but it the familywise error rate control is guaranteed only if the hypotheses tested in each node of the graph with only leaf nodes as offspring is precisely the intersection hypothesis of its child nodes. When doing the inheritance procedure in combination with the global test, this condition is fulfilled if the set of covariates at each node with only leaf nodes as offspring is precisely the union of the sets of covariates of its offspring leaf nodes. This condition is fulfilled for the `tree` graph above, but if we had set `level1 <- as.list(LETTERS[19])`:, the node `hij` contains a covariate (`J`) that is not present in any of its child nodes, so that the condition for the Shaffer improvement is not fulfilled, and setting `Shaffer = TRUE` does not control the familywise error rate. If *test* is a *gt.object* the procedure check if structure of *sets* allows for a Shaffer improvement, and sets *Shaffer* to the correct default. In other cases, checking the validity of the Shaffer improvement is left to the user. Note that setting `Shaffer = TRUE` always gives a correct procedure.

The tree structure of the hypotheses may be fixed a priori, based on the prior knowledge rather than on the data. However, in some situations a data-driven definition of the structure is allowed. Meinshausen (2008) suggests to use a hierarchical clustering method using as distance matrix based on the (correlation) distance between explanatory covariates. This is valid for the global test, and may in some cases also be valid if other tests are performed.

In `inheritance`, the tree-structured graph *sets* can be an object of class *hclust* or *dendrogram*. If *sets* is missing and *test* is a *gt.object* the structure is derived from the structure of *test*.

```
> hc <- hclust(dist(t(X)))
> resHC <- inheritance(res, hc)
> resHC
```

```
                inheritance  p-value Statistic Expected Std.dev #Cov
O[2[2[2[2[2[2:F    1.00e+00 3.51e-01     4.856     5.26    7.24    1
O[2[2[1            3.69e-02 8.21e-03    24.238     5.26    5.36    2
O                  7.34e-06 7.34e-06    24.327     5.26    2.79   10
O[2[2[1[1:A        3.69e-02 2.00e-03    41.998     5.26    7.24    1
O[1                5.24e-05 1.67e-05    53.142     5.26    5.94    2
O[2[2[1[2:H        1.00e+00 6.92e-01     0.895     5.26    7.24    1
```

```
O[1[1:B            5.24e-05 5.72e-06    69.036     5.26    7.24    1
O[2[2[2            9.29e-01 4.89e-01     4.500     5.26    3.91    4
O[1[2:C            2.03e-02 6.47e-03    34.494     5.26    7.24    1
O[2[2[2[1:J        1.00e+00 7.51e-01     0.573     5.26    7.24    1
O[2                3.19e-02 3.19e-02    10.841     5.26    2.67    8
O[2[2[2[2          9.29e-01 2.63e-01     7.092     5.26    4.23    3
O[2[1              9.25e-01 2.69e-01     6.788     5.26    5.76    2
O[2[2[2[2[1:D      9.29e-01 1.07e-01    13.754     5.26    7.24    1
O[2[1[1:G          9.29e-01 2.70e-01     6.704     5.26    7.24    1
O[2[2[2[2[2        1.00e+00 6.72e-01     2.110     5.26    5.45    2
O[2[1[2:I          9.29e-01 2.62e-01     6.931     5.26    7.24    1
O[2[2[2[2[2[1:E    1.00e+00 8.30e-01     0.263     5.26    7.24    1
O[2[2              3.69e-02 2.62e-02    12.506     5.26    3.15    6
```

It is a property of both the inheritance and the focus level method, that the adjusted p-value of a node can never be smaller than a p-value of an ancestor node. The significant graph at a certain significance level is therefore always a coherent graph, which always contains all ancestor nodes of any rejected node. Such a graph can be succinctly summarized by reporting only its leaf nodes. This can be done using the `leafNodes` function.

```
> leafNodes(resI)
```

```
  inheritance  p-value Statistic Expected Std.dev #Cov
a    7.06e-03 2.00e-03      42.0     5.26    7.24    1
b    2.02e-05 5.72e-06      69.0     5.26    7.24    1
c    2.49e-02 6.47e-03      34.5     5.26    7.24    1
```

```
> leafNodes(resHC)
```

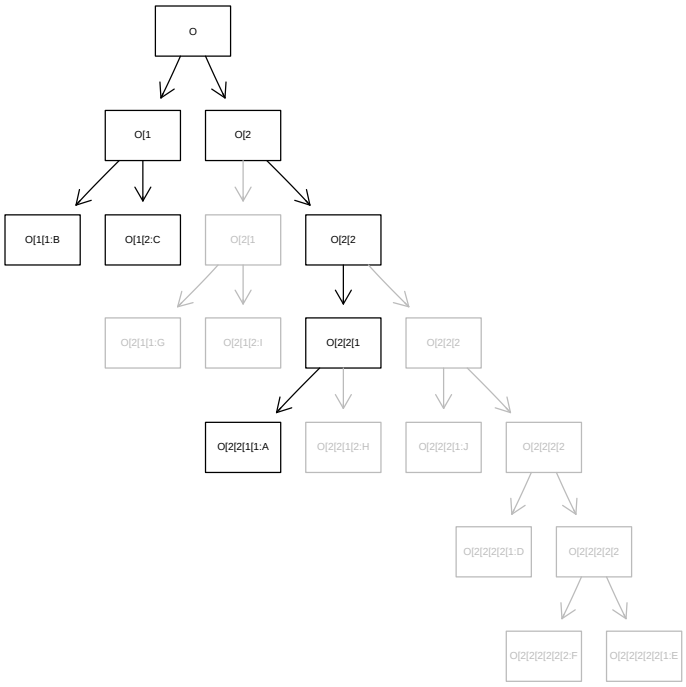
```
            inheritance  p-value Statistic Expected Std.dev #Cov
O[2[2[1[1:A    3.69e-02 2.00e-03      42.0     5.26    7.24    1
O[1[1:B        5.24e-05 5.72e-06      69.0     5.26    7.24    1
O[1[2:C        2.03e-02 6.47e-03      34.5     5.26    7.24    1
```

The *alpha* argument of the `leafNodes` function can be used to specify the rejection threshold for the familywise error of the significant graph.

Like for `focusLevel`, the `draw` can be used to visualize the test result as a graph: However, in most cases the `covariates` function does a better graphical job. `covariates` performs `hclust` on the covariates and calls the `inheritance` function using this data-driven structure.

```
> covariates(res)
```

```
> draw(resHC, names = TRUE)
```



Chapter 3

Gene Set Testing

3.1 Introduction

One important application of the global test is in gene set testing in gene expression microarray data (Goeman et al., 2004, 2005). Such data consist of simultaneous gene expression measurements of many thousands of probes across the genome, performed for a number of biological samples. The typical goal of a microarray experiment is to find associations between the expression of genes and a phenotype variable.

Gene set testing is a common denominator for a type of analysis for microarray data that takes together groups of genes that have a common annotation, e.g. which are all annotated to the same Gene Ontology term, which are all members of the same KEGG pathway, or which have a similar chromosomal location. Gene set testing methods test such gene sets together to investigate whether the genes in the gene set have a higher association with the response than expected by chance. These methods provide a single p-value for the gene set, rather than a p-value for each gene.

The global test is well suited for gene set testing; in fact, the global test was initially designed specifically with this application in mind (Goeman et al., 2004). The model that the global test uses for gene set testing is a regression model, such as might also be used to predict the response based on the gene expression measurements: in this model the gene expression measurements correspond to the covariates and the phenotype corresponds to the response. The null hypothesis that the global test tests is the null hypotheses that all regression coefficients of all the genes in the gene set are zero, i.e. the genes in the gene set have no predictive ability for predicting the response. The global test can therefore be seen as a method that looks for differentially expressed gene sets.

The global test tests gene sets in a single step, based on the full data, without an intermediate step of finding individual differentially expressed genes. In the classification scheme for gene set testing methods of Goeman and Bühlmann (2007), the global test is a *self-contained* method rather than a *competitive* one: it tests the null hypothesis that no gene in the gene set is associated with the phenotype rather than the null hypothesis that the genes in the gene set are not more associated with the phenotype

than random genes on the microarray. The latter approach is followed by enrichment methods such as GSEA and methods based on Fisher's exact test. The global test is also a *subject-sampling* rather than a *gene-sampling* method. This means that when determining whether the genes in the gene set have a higher association with the phenotype than expected by chance, the method looks at the random biological variation between subjects, rather than comparing the gene set with random sets of genes. The latter approach is used by gene set testing methods based on Fisher's exact test. Unlike the validity of gene-sampling methods, the validity of subject-sampling methods does not depend on the unrealistic assumption that gene expression measurements are independent.

As shown by Goeman et al. (2006), the global test is designed to have optimal power in the situation in which the gene set has many small non-zero regression coefficients. This means that the test is especially directed to find gene sets for which many genes are associated with the phenotype in a small way. This behavior is appropriate for gene set testing, because the situation that many genes are associated with the phenotype is usually the most interesting from a gene set perspective. Still, it is true that the null hypotheses that the global test tests is false even if only a single gene in the gene set is associated with the phenotype; especially smaller gene sets may therefore become significant as a result of only a single significant gene. However, because the test is directed especially against the alternative that there are many associated genes, such examples are rare among larger gene sets.

3.2 Data format

The globaltest package uses the usual statistical orientation of data matrices in which the columns of the data matrix correspond to covariates, and the rows of the data matrix correspond to subjects. In gene set testing and in other genomics applications it is more common to use the reverse orientation, in which the columns of the data matrix correspond to the subjects and the rows to the covariates. The `gt.options` function can be used to change the default orientation expected by `gt` for the *alternative* argument.

```
> gt.options(transpose = TRUE)
```

Note that this option is only relevant if *alternative* is given as a matrix. A *formula* or *ExpressionSet* input (Section 3.2.1) input for *alternative* is automatically interpreted correctly.

3.2.1 Using *ExpressionSet* data

We illustrate gene set testing using the Golub et al. (1999) data set, a famous data set which was one of the first to use microarray data in a classification context. This dataset is available from bioconductor as the *golubEsets* package. We load the `Golub_Train` data set, consisting of 38 Leukemia patients for which 7129 gene expression measurements were taken.

```
> library(golubEsets)
> data(Golub_Train)
```

The Golub_Train data are in *ExpressionSet* format, which is the standard format in bioconductor for storing gene expression data. The *ExpressionSet* objects contain the gene expression data, phenotypic data, and annotation information about the genes and the experiment, all in the same R object. The data have to be properly normalized and log- or otherwise transformed, as usual in microarray data. We keep the normalization simple and use only *vsn*.

```
> library(vsn)
> exprs(Golub_Train) <- exprs(vsn2(Golub_Train))
```

The phenotype of interest is the leukemia subtype, coded as the variable ALL.AML, with values "ALL" and "AML", in pData(Golub_Train). It is generally a good idea to start by testing the overall expression profile to see whether that is notably different between AML and ALL patients. We supply the *ExpressionSet* Golub_Train in the *alternative* argument of gt. Because the *alternative* argument is of class *ExpressionSet*, the function now uses t(exprs(Golub_Train)) as the *alternative* argument and pData(Golub_Train) as the *data* argument.

```
> gt(ALL.AML, Golub_Train)

   p-value Statistic Expected Std.dev #Cov
1 1.78e-11      10.1     2.70   0.581 7129
```

From the test result we conclude that the overall expression profile of ALL patients and AML patients differs markedly in this experiment. This is not very surprising, as this data set has been used in many papers as an example of a data set that can be classified very easily. From this result we may expect to find many genes and gene sets to be differentially expressed.

If the overall test is not significant or only marginally significant, it can be difficult to find many genes or pathways that are differentially expressed. In this case it is usually not a good idea to perform a broad untargeted data mining type analysis of the data, e.g. by testing complete pathway databases, because it is likely that in this case the signal of the genes and gene sets that are differentially expressed is drowned in the noise of the genes that are not differentially expressed. A more targeted approach focussed on a limited number of candidate gene sets may be more opportune in such a situation.

Adjustment of the test result for confounders such as batch effects, clinical or phenotype covariates can be specified by specifying these variables as covariates under the null hypothesis, as described in Section 2.1.3. When using *ExpressionSet* data, the easiest way to do this is with a *formula*. The terms of such a *formula* are automatically interpreted in terms of the pData slot of the *ExpressionSet*. Missing data ar not allowed in phenotype variables, so we illustrate the adjustment for confounders by correcting for the data source in the Golub data (the DFCI and CALGB centers)

```
> gt(ALL.AML ~ Source, Golub_Train)

  p-value Statistic Expected Std.dev #Cov
1       1 -2.43e-15     2.93   0.525 7129
```

In this specific case we see that the association between gene expression and disease subtype is completely confounded by the source variable. In fact, all ALL patients came from DFCI, and all AML patients from CALGB. In this case we cannot distinguish between the effects of disease subtype from the center effects: the design of this study is, unfortunately, broken.

3.2.2 Other input formats

Alternatively, the formula or matrix-based input described in Section 2.1 may also be used instead of the *ExpressionSet*-based one. For matrix-based input, `gt` expects the usual statistical data-format in which the subjects correspond to the rows of the data matrix and the covariates (probes or genes) are the columns. The option *transpose* in `gt.options` can be used to change this. Setting

```
> gt.options(transpose = TRUE)
```

changes the default behavior of `gt` to expect the transposed format that is usual in genomics, with the rows of the data matrix corresponding to the genes and the columns to the subjects.

The `gtKEGG`, `gtGO` and `gtBroad` functions (Section 3.3) always expect the genomics data format rather than the usual statistical format.

3.2.3 The *trim* option

A second useful option to set when doing gene set testing is the *trim* option. This option governs the way `gt` handles covariate names that appear in the *subsets* argument, but are not present in the expression data matrix. The default behavior of `gt` is to return an error when this happens. However, in gene set testing covariates may easily be missing from the expression data, for example because the subsets are based on the annotation of the complete microarray, while some genes have been removed from the expression data matrix, perhaps due to poor measurement quality. Setting

```
> gt.options(trim = TRUE)
```

makes `gt` silently remove such missing covariates from the *subsets* argument.

Additionally, if `trim = TRUE`, duplicate covariate names in *subsets* are automatically removed.

3.3 Testing gene set databases

The most common approach to gene set testing is to test gene sets from public databases. The globaltest package provides utility functions for three such databases: Gene Ontology, KEGG and the pathway databases maintained by the Broad Institute. In all cases, these functions make heavy use of the annotation packages available in Bioconductor. If the microarray that was used does not have an annotation package, the Entrez-based organism annotation packages (e.g. *org.Hs.eg.db* for human) can be used instead.

3.3.1 KEGG

The function `gtKEGG` can be used to test KEGG terms. To test a single KEGG id, e.g. cell cycle (KEGG id `04110`), use

```
> gtKEGG(ALL.AML, Golub_Train, id = "04110")

            alias  p-value Statistic Expected Std.dev #Cov
04110 Cell cycle 4.69e-08      12.1     2.70   0.878  109
```

The function automatically finds the right KEGG information from the *KEGG.db* package, and the right set of genes belonging to the KEGG id from the annotation package of the *hu6800* Affymetrix chip; the reference to this annotation package is contained in the `Golub_Train` *ExpressionSet* object. If *ExpressionSet* objects are not used, the name of the annotation package can be supplied in the *annotation* argument of `gtKEGG`.

Annotation packages are not always available for all microarray types. Therefore, a general Entrez-based annotation package is available for many organisms, e.g. *org.Hs.eg.db* for human. See `www.bioconductor.org` for the names of the organism specific packages. This general entrez-based annotation package may be substituted for a specific probe annotation package if a mapping from probe(set) ids to Entrez is given (as a list or as a vector) in the *probe2entrez* argument. For the Golub data we find such a mapping in the *hu6800.db* package.

```
> eg <- as.list(hu6800ENTREZID)
> gtKEGG(ALL.AML, Golub_Train, id = "04110", probe2entrez = eg,
      annotation = "org.Hs.eg.db")

            alias  p-value Statistic Expected Std.dev #Cov
04110 Cell cycle 4.69e-08      12.1     2.70   0.878  109
```

If more than one KEGG id is tested, multiple testing corrected p-values are automatically provided. The default multiple testing method is Holm's, but others are available through the *multtest* argument. See also the `p.adjust` function, described in Section 2.3.2. The results are sorted to increasing p-values (using the `sort` method), unless the *sort* argument of `gtKEGG` is set to `FALSE`.

```
> gtKEGG(ALL.AML, Golub_Train, id = c("04110", "04210"), multtest = "BH")

             BH      alias  p-value Statistic Expected Std.dev #Cov
04110 9.38e-08 Cell cycle 4.69e-08     12.15     2.70   0.878  109
04210 5.73e-05  Apoptosis 5.73e-05      9.61     2.70   0.987   79
```

If the *id* argument is not specified, the function `gtKEGG` will test all KEGG pathways.

```
> gtKEGG(ALL.AML, Golub_Train)
```

3.3.2 Gene Ontology

To test Gene Ontology terms the special function `gtGO` is available. This function accepts the same arguments as `gt`, except the *subsets* argument, which is replaced by a collection of options to create gene sets from Gene Ontology. To test a single gene ontology term, e.g. cell cycle (`GO:0007049`), we say

```
> gtGO(ALL.AML, Golub_Train, id = "GO:0007049")

                alias  p-value Statistic Expected Std.dev #Cov
GO:0007049 cell cycle 3.35e-09      12.3      2.7   0.753  495
```

The function automatically finds the right Gene Ontology information from the *GO.db* package, and the right set of genes belonging to the gene ontology term from the annotation package of the *hu6800* Affymetrix chip; the reference to this annotation package is contained in the `Golub_Train` *ExpressionSet* object. If *ExpressionSet* objects are not used, the name of the annotation package can be supplied in the *annotation* argument of `gtGO`.

Annotation packages are not always available for all microarray types. Therefore, a general Entrez-based annotation package is available for many organisms, e.g. *org.Hs.eg.db* for human. See `www.bioconductor.org` for the names of the organism specific packages. This general entrez-based annotation package may be substituted for a specific probe annotation package if a mapping from probe(set) ids to Entrez is given (as a list or as a vector) in the *probe2entrez* argument. For the Golub data we find such a mapping in the *hu6800.db* package.

```
> eg <- as.list(hu6800ENTREZID)
> gtGO(ALL.AML, Golub_Train, id = "GO:0007049", probe2entrez = eg,
      annotation = "org.Hs.eg")

                alias  p-value Statistic Expected Std.dev #Cov
GO:0007049 cell cycle 3.35e-09      12.3     2.70   0.753  495
```

It is also possible to test all terms in one or more of the three ontologies: Biological Process (BP), Molecular Function (MF) and Cellular component (CC). A minimum and/or a maximum number of genes may be specified for each term.

```
> gtGO(ALL.AML, Golub_Train, ontology = "BP", minsize = 10, maxsize = 500)
```

If more than one gene ontology term is tested, multiple testing corrected p-values are automatically provided. The default multiple testing method is Holm's, but others are available through the *multtest* argument. See also the `p.adjust` function, described in Section 2.3.2. The results are sorted to increasing p-values (using the `sort` method), unless the *sort* argument of `gtGO` is set to `FALSE`.

```
> gtGO(ALL.AML, Golub_Train, id = c("GO:0007049", "GO:0006915"),
      multtest = "BH")

                 BH      alias  p-value Statistic Expected Std.dev #Cov
GO:0006915 5.02e-12  apoptosis 2.51e-12      11.2      2.7   0.679  692
GO:0007049 3.35e-09 cell cycle 3.35e-09      12.3      2.7   0.753  495
```

A multiple testing method that is very suitable for Gene Ontology is the focus level method, described in more detail in Section 2.3.3. This multiple testing method presents a coherent significant subgraph of the Gene Ontology graph. This is a relatively computationally intensive method. To keep this vignette light, we shall only demonstrate the focus level method on the subgraph of "cell cycle" GO term and all its descendants.

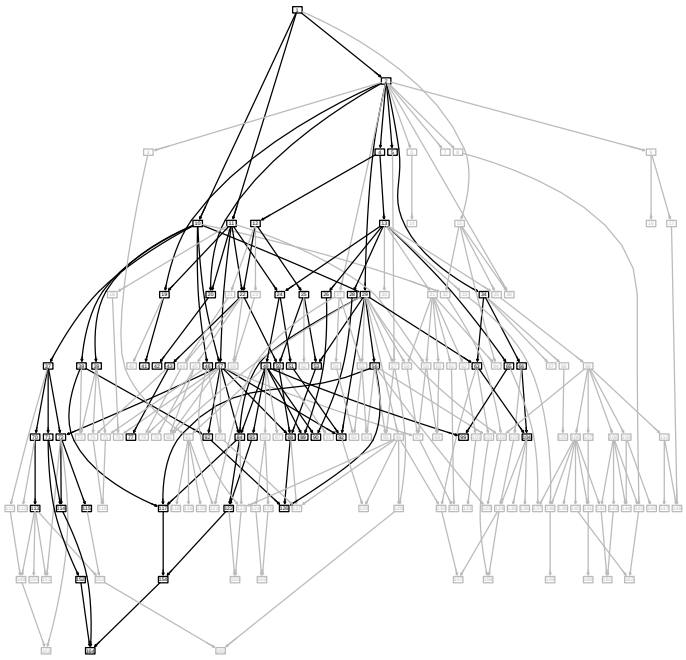
```
> descendants <- get("GO:0007049", GOBPOFFSPRING)
> res <- gtGO(ALL.AML, Golub_Train, id = c("GO:0007049", descendants),
      multtest = "focus")
> leafNodes(res)
```

```
           focuslevel                                          alias  p-value
GO:0007050   1.93e-07                                cell cycle arrest 3.51e-09
GO:0045749   3.93e-04 negative regulation of S phase of mitotic ... 1.58e-07
GO:0000077   5.80e-04                            DNA damage checkpoint 9.67e-06
GO:0031575   1.37e-03                      G1/S transition checkpoint 1.58e-05
GO:0051436   3.81e-03 negative regulation of ubiquitin-protein l... 6.93e-05
GO:0010824   6.63e-03           regulation of centrosome duplication 1.21e-04
GO:0000132   7.17e-03  establishment of mitotic spindle orientation 1.05e-04
GO:0007096   7.88e-03                regulation of exit from mitosis 1.00e-04
GO:0000083   1.22e-02 regulation of transcription involved in G1... 2.25e-04
GO:0000090   2.20e-02                                 mitotic anaphase 4.07e-04
GO:0000093   2.20e-02                                mitotic telophase 4.07e-04
GO:0000089   2.50e-02                                mitotic metaphase 4.26e-04
GO:0045787   2.68e-02              positive regulation of cell cycle 5.16e-04
GO:0000079   2.71e-02 regulation of cyclin-dependent protein kin... 5.21e-04
GO:0007094   3.12e-02 mitotic cell cycle spindle assembly checkp... 5.99e-04
           Statistic Expected Std.dev #Cov
GO:0007050     17.53     2.70   1.106   86
GO:0045749     21.60     2.70   1.688   14
GO:0000077     11.84     2.70   1.274   24
GO:0031575     12.36     2.70   1.381   19
GO:0051436     16.56     2.70   1.708   44
GO:0010824     17.83     2.70   2.061    5
GO:0000132     31.14     2.70   3.389    2
GO:0007096     15.74     2.70   1.892    6
GO:0000083     15.10     2.70   1.965   10
GO:0000090     29.66     2.70   3.770    1
GO:0000093     29.66     2.70   3.770    1
GO:0000089     27.46     2.70   3.493    2
GO:0045787      7.85     2.70   1.026   50
GO:0000079      7.49     2.70   0.996   36
GO:0007094     11.02     2.70   1.544    9
```

The leaf nodes can be seen as a summary of the significant GO terms: they present the most specific terms that have been declared significant at a specified significance level

alpha (default 0.05). The graph can be drawn using the `draw` function. In the interactive mode of this function, click on the nodes to see the GO id and term. The default of this function is to draw the full graph, with the non-significant nodes greyed out. It is also possible to only draw the significant graph by setting the *sign.only* argument to `TRUE`. The draw function returns a legend to the graph, relating the numbers appearing in the plot to the GO terms. This is useful when using `draw` in non-interactive mode

```
> draw(res, interactive = TRUE)
> legend <- draw(res)
```



3.3.3 The Broad gene sets

A third frequently used database is the collection of curated gene sets maintained by the Broad institute. The sets are only available after registration at `http://www.broad.mit.edu/gsea/downloads.jsp#msigdb`. To use the Broad gene sets, download the file `msigdb_v.2.5.xml`, which contains all sets. A convenient function to read the xml file into `R` is provided in the `getBroadSets` function from the *GSEABase* package. Once downloaded and read, the `gtBroad` function can be used to analyze these gene sets using the global test.

```
> broad <- getBroadSets("your/path/to/msigdb_v.2.5.xml")
```

The examples in this vignette are displayed without results, because we cannot include the `msigdb_v.2.5.xml` file in the *globaltest* package.

To test a single Broad set, e.g. the chromosomal location chr5q33, use

```
> gtBroad(ALL.AML, Golub_Train, id = "chr5q33", collection = broad)
```

The function automatically maps the gene set to the probe identifiers from the annotation package of the *hu6800* Affymetrix chip; the reference to this annotation package is contained in the `Golub_Train` *ExpressionSet* object. If *ExpressionSet* objects are not used, the name of the annotation package can be supplied in the *annotation* argument of `gtBroad`.

Annotation packages are not always available for all microarray types. Therefore, a general Entrez-based annotation package is available for many organisms. This general annotation package may be substituted for a specific annotation package if a mapping from probe(set) ids to Entrez is given (as a list or as a vector). For the Golub data we use the mapping from the *hu6800.db* package to obtain this mapping.

```
> eg <- as.list(hu6800ENTREZID)
> gtBroad(ALL.AML, Golub_Train, id = "chr5q33", collection = broad,
      probe2entrez = eg, annotation = "org.Hs.eg.db")
```

See `www.bioconductor.org` for the names of the organism specific packages.

If more than one Broad set is tested, multiple testing corrected p-values are automatically provided. The default multiple testing method is Holm's, but others are available through the *multtest* argument. See also the `p.adjust` function, described in Section 2.3.2. The results are sorted to increasing p-values (using the `sort` method), unless the *sort* argument of `gtBroad` is set to `FALSE`.

```
> gtBroad(ALL.AML, Golub_Train, id = c("chr5q33", "chr5q34"), multtest = "BH",
      collection = broad)
```

The broad collection contains four categories

c1 positional gene sets

c2 curated gene sets

c3 motif gene sets

c4 computational gene sets

c5 GO gene sets

To test all gene sets from a certain category, use

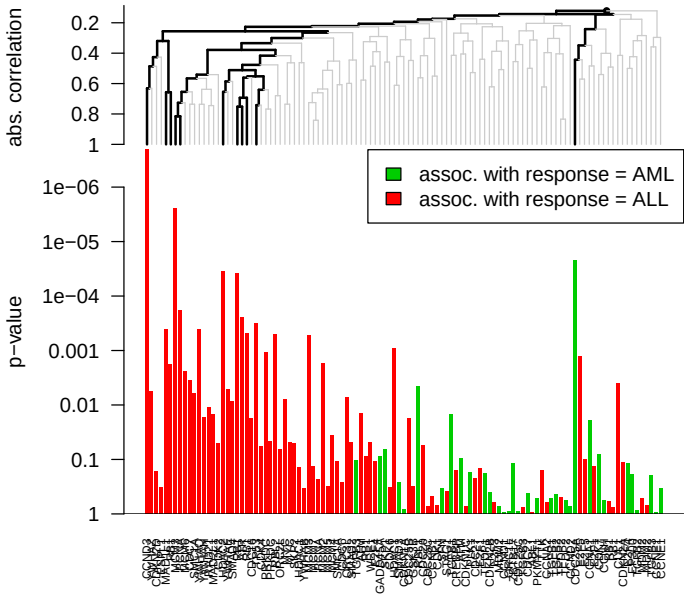
```
> gtBroad(ALL.AML, Golub_Train, category = "c1", collection = broad)
```

3.4 Gene and sample plots

3.4.1 Visualizing features

The covariate (or "features") plot may be used to great effect for investigating to which individual probes or genes or to which subsets of the gene set a significant result for a gene set may be attributed. The details of the `features` plot are described in Section

```
> res <- gtKEGG(ALL.AML, Golub_Train, id = "04110")
> features(res, alias = hu6800SYMBOL)
```



2.2.1. The *alias* argument is useful to replace the probe identifiers with more familiar gene symbols.

The black line in the `features` plot represents the significant subgraph of the clustering tree. To find the leaf nodes that characterize the graph, use the function `leafNodes`.

```
> ft <- features(res, alias = hu6800SYMBOL)
> leafNodes(ft)
```

```
                                                    alias inheritance
O[1[1[1[1[1[1[1[1[1[1[1[1[1[1[1[1:M92287_at          CCND3    0.000116
O[1[1[1[1[1[1[1[1[1[1[1[1[1[2[1:U33822_at          MAD1L1    0.008918
```

```
O[1[1[1[1[1[1[1[1[1[1[2[2:L49219_f_at                    RB1    0.038838
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[1[1[1[1[1[1[1:D38073_at    MCM3    0.000683
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[1[1[1[1[1[2:M15796_at      PCNA    0.006328
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[1[1[1:U31814_at     HDAC2    0.004149
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[1[1:L41870_at    RB1    0.002170
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[1[2:U49844_at    ATR    0.013903
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[2:L49229_f_at    RB1    0.017646
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[2[1[1:M22898_at     TP53    0.036909
O[1[2[1[1[1[1:M81933_at                                  CDC25A    0.004907
                                                          p-value Statistic
O[1[1[1[1[1[1[1[1[1[1[1[1[1[1[1:M92287_at                2.06e-07      53.2
O[1[1[1[1[1[1[1[1[1[1[2[1:U33822_at                      4.07e-04      29.7
O[1[1[1[1[1[1[1[1[1[1[2[2:L49219_f_at                    1.77e-03      24.1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[1[1[1[1[1[1:D38073_at      2.48e-06      46.4
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[1[1[1[1[1[2:M15796_at      1.85e-04      32.5
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[1[1[1:U31814_at      3.58e-05      38.2
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[1[1:L41870_at  3.82e-05      38.0
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[1[2:U49844_at  2.45e-04      31.5
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[2:L49229_f_at  4.83e-04      29.0
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[2[1[1:M22898_at    3.16e-04      30.6
O[1[2[1[1[1[1:M81933_at                                  2.18e-05      39.8
                                                         Expected Std.dev #Cov
O[1[1[1[1[1[1[1[1[1[1[1[1[1[1[1:M92287_at                    2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1:U33822_at                          2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[2:L49219_f_at                        2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[1[1[1[1[1[1:D38073_at          2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[1[1[1[1[1[2:M15796_at          2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[1[1[1:U31814_at          2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[1[1:L41870_at      2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[1[2:U49844_at      2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[1[1[2:L49229_f_at      2.70    3.77    1
O[1[1[1[1[1[1[1[1[1[1[2[1[1[1[2[1[1[2[2[1[1:M22898_at        2.70    3.77    1
O[1[2[1[1[1[1:M81933_at                                      2.70    3.77    1
```

It may happen that the leaf nodes of the significant graph are not individual features, but sets of features higher up in the clustering graph. Use the `subsets` method to find which features belong to such a node.

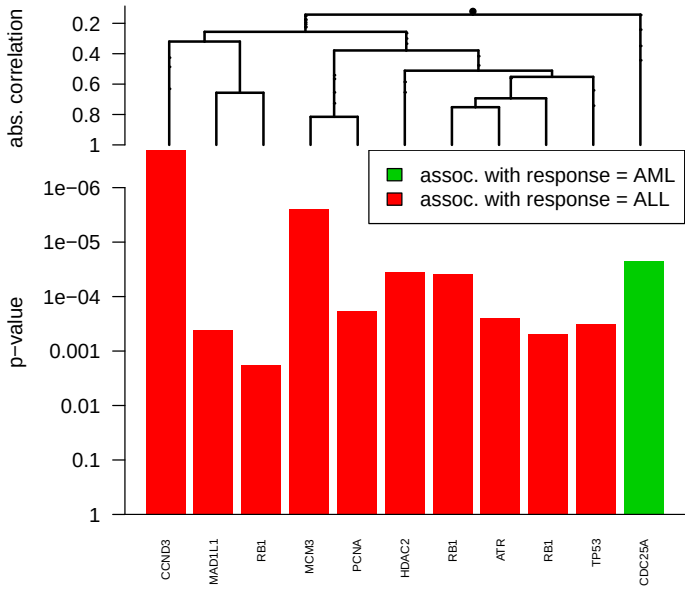
```
> subsets(leafNodes(ft))
```

It is possible to only plot the significant subtree with the *zoom* argument. This is especially useful if the set of features is large.

When testing many GO or KEGG terms it can be convenient to make features plots for all tested gene sets at once, writing all plots to a pdf.

```
> res_all <- gtKEGG(ALL.AML, Golub_Train)
> features(res_all[1:5], pdf = "KEGGcov.pdf", alias = hu6800SYMBOL)
```

```
> res <- gtKEGG(ALL.AML, Golub_Train, id = "04110")
> features(res, alias = hu6800SYMBOL, zoom = TRUE)
```

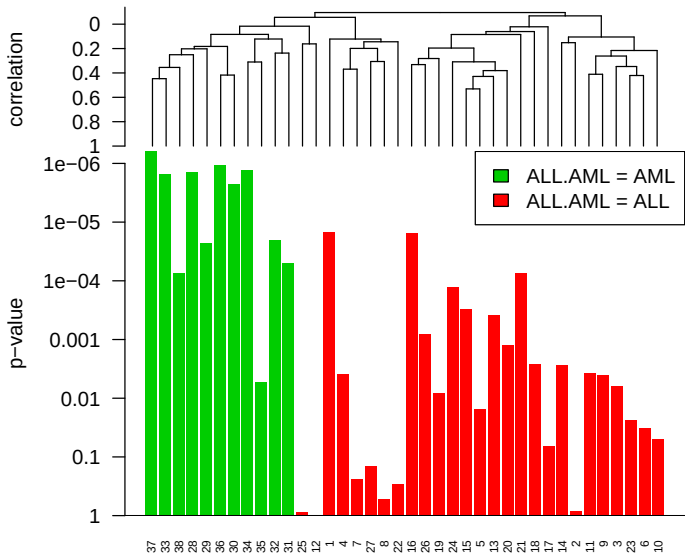


3.4.2 Visualizing subjects

Similarly, the subjects plot, described in Section 2.2.2, can be used to investigate which subjects are similar in terms of their expression signature to other subjects with the same response variable, and which deviate from that pattern. In the `subjects` diagnostic plot, the subjects associated with strong evidence for the association between the response and the gene expression profile of the pathway have low p-values (tall bars), whereas the subjects with high p-values have weak or even contrary evidence. The most interesting subjects plot to look at is usually the subjects plot for the global test on all genes. From the figure, in this case, we note that the expression profile of the AML subjects seems more homogeneous than that of the ALL subjects: the latter group tends to be less coherent overall, and to contain more outlying subjects. Just as with the `covariates` plot, `subjects` plots can be called on many gene sets at once, e.g. the top 25 pathways, and the results written to a pdf file.

```
> res_all <- gtKEGG(ALL.AML, Golub_Train)
> subjects(res_all[1:25], pdf = "KEGGsubj.pdf")
```

```
> res <- gt(ALL.AML, Golub_Train)
> subjects(res)
```



3.5 Survival data

The examples in this chapter so far were all in a classification context, in which the response category had two possible values, and the logistic regression model was used. The *globaltest* package is not limited to this response type, but can also handle multi-category response (using a multinomial logistic regression model), continuous response (using a linear regresseion model), count data (using a Poisson regression model), and survival data (using the Cox proportional hazards model). See section 2.1.7 for more details.

The multi-category, linear and count data versions are called in exactly the same way as the two-category one. The `gt` function will try to determine the model from the input, but the user can override any automatic choice by specifying the *model* argument.

For survival data, the input format is similar to the one used by the *survival* package. In the `michigan` data set (Beer et al., 2002) from the *lungExpression* package, for example, the survival time is coded in a time variable `TIME..months.`, and a status variable `death`, for which 1 indicates that the patient passed away at the recorded time point, and 0 that the patient was withdrawn alive. To test for an overall association between the gene expression profile and survival, we test

```
> library(lungExpression)
> data(michigan)
> gt(Surv(TIME..months., death == 1), michigan)

  p-value Statistic Expected Std.dev #Cov
1   0.188      1.53     1.16   0.417 3171
```

3.6 Comparative proportions

In some cases it can be of interest not only to know whether a certain gene set is significantly associated with a phenotype, but also whether it is exceptionally associated with the phenotype for a gene set of its size in the data set under study. This is a so-called competitive view on gene set testing. See Goeman and Bühlmann (2007) for issues involved with this competitive view.

It is possible to use *globaltest* for such competitive gene set exploration using the function `comparative`. For each gene set tested, this function calculates the proportion of randomly sampled gene sets of the same size as the tested gene set that has an equal or larger global test p-value. This comparative proportion can be used as a diagnostic for the test results. Gene sets with small comparative proportions are exceptionally significant in comparison to a random gene set of its size in the data set. The comparative proportion is a diagnostic that conveys additional information. It should not be interpreted as a p-value in the usual sense.

```
> res <- gtKEGG(ALL.AML, Golub_Train, id = "04110")
> comparative(res)

           alias comparative  p-value Statistic Expected Std.dev #Cov
04110 Cell cycle       0.209 4.69e-08      12.1     2.70   0.878  109
```

The number of random gene sets of each size that are sampled can be controlled with the argument *N* (default 1000). The argument *zscores* (default: `TRUE`) controls whether the comparison between the test results of the gene set and its random competitors is based on the p-values or on the z-scores of the test.

Bibliography

Beer, D. G., Kardia, S. L. R., Huang, C. C., Giordano, T. J., Levin, A. M., Misek, D. E., Lin, L., Chen, G. A., Gharib, T. G., Thomas, D. G., Lizyness, M. L., Kuick, R., Hayasaka, S., Taylor, J. M. G., Iannettoni, M. D., Orringer, M. B., and Hanash, S. (2002). Gene-expression profiles predict survival of patients with lung adenocarcinoma. *Nature Medicine*, 8(8):816–824.

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B-Methodological*, 57(1):289–300.

Benjamini, Y. and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 29(4):1165–1188.

Goeman, J. and Bühlmann, P. (2007). Alalyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics*, 23(8):980–987.

Goeman, J. J., Finos, L., and Van Houwelingen, H. C. (2009). Testing against a high-dimensional alternative in generalized linear models. Unpublished preprint.

Goeman, J. J. and Mansmann, U. (2008). Multiple testing on the directed acyclic graph of gene ontology. *Bioinformatics*, 24(4):537–544.

Goeman, J. J., Oosting, J., Cleton-Jansen, A. M., Anninga, J. K., and van Houwelingen, J. C. (2005). Testing association of a pathway with survival using gene expression data. *Bioinformatics*, 21(9):1950–1957.

Goeman, J. J., van de Geer, S. A., de Kort, F., and van Houwelingen, J. C. (2004). A global test for groups of genes: testing association with a clinical outcome. *Bioinformatics*, 20(1):93–99.

Goeman, J. J., van de Geer, S. A., and van Houwelingen, J. C. (2006). Testing against a high-dimensional alternative. *Journal of the Royal Statistical Society Series B-Statistical Methodology*, 68(3):477–493.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70.

Meinshausen, N. (2008). Hierarchical testing of variable importance. *Biometrika*, 95(2):265–278.