YAMAS
–
Yet Another Meta-Analysis Software

Christian Meesters
Markus Leber
Christine Herold
Manuel Mattheisen
Dmitriy Drichel
André Lacour
Tim Becker

Deutsches Zentrum für Neurodegenerative Erkrankungen (DZNE)
German Centre for Neurodegenerative Diseases
Outstation Bonn

Institute for Medical Biometrics, Informatics and Epidemiology (IMBIE)
University of Bonn

Bonn, August 1, 2012

Contents

Chapter 1

Introduction

Welcome to YAMAS (**Y**et **A**nother **M**eta-**A**nalysis **S**oftware)!

YAMAS tries to fulfill various meta-analysis needs with regard to genome-wide association studies. There are other softwares around, but we hope with YAMAS more features are provided and more possible pitfalls avoided.

This manual should provide you with a comprehensive overview of the software. Still, if after referring to this manual a nagging question is left (or, if you discover a bug), you are most welcome to write us ✉Tim.Becker@dzne.de and ✉meesters@aesku-kipp.com[1] (please put both emails in the address field of your mail client) in case of any nagging question.

Note that you can **click on every link to jump within this manual**.

In order to get started we recommend reading at least the following chapter. For the rest skip whatever you like, but be aware that you might miss essential information which you need to exploit the full power of YAMAS.

Enjoy!

[1] I left the institute shortly after writing the software and am now to find at the AESKU.KIPP-Institute in Wendelsheim, Germany (see `www.aesku-kipp.com`).

Chapter 2

Using YAMAS

First things first: This chapter shows several examples on the shell. Parameters given in angular brackets (e. g. `[parameter]`) are optional, required arguments are shown as `<arguments>`.

Sometimes commands did not fit in one line. In this case you might see a backslash (\) to indicate a line break. You may or may not copy this to your command shell - it doesn't matter for all shells known to us.

2.1 Defining Input

Input may be defined using the command line or a configuration file. However, not all arguments can be given using both interfaces.

2.1.1 The Commandline

Command line arguments are given as follows:

```
$ yamas -t 4
$ # or
$ yamas --threads 4
```

Options usually come with a short (-) and a long (--), more verbose, version. You may choose, they are equivalent. However, some options are short or long, only. The following table shows a complete list of options.

And you may ask YAMAS for help, if you like:

```
$ yamas -h
$ # or
$ yamas --help
```

A full list of possible command line options is given along with their defaults.

Table 2.1: *List of available command line options.* A detailed description for some is given below.

Option	Description	Reference
`-t, --threads`	gives the number of threads to use for parallel algorithms	see paragraph 2.1.1.1
`-c, --config`	specifies path to the configuration file	see paragraph 2.1.4
`-l, --logfile`	specifies name of the log file YAMAS produces – this is also the major output file	see paragraph 2.1.1.3
`-a, --algo`	specifies the algorithm to perform	see paragraph 2.1.1.2
`--odds_ratio`	whether or not YAMAS is dealing with odds ratios, default is no	—
`-v, --verbose`	if set, YAMAS will behave a little more verbose	see paragraph 2.1.1.5
`--cl`	set confidence level for confidence intervals (default value: 0.95)	see paragraph 2.2.3
`-p, --pthreshold`	only markers with combined P-vlalues $\leq$ this threshold will be written to the log file	—
`--r2threshold`	only proxy SNPs with an mutual $r^2 \geq$ this threshold will be considered	see paragraph 3.5.1
`--chromosome`	can be used to select a specific chromosome to be analysed, only	see paragraph 2.1.1.4
`--assocfilename`	if set, a basic association test is performed	see paragraph 3.1
`-m, --missing`	can be used to alter the missing allele for an association test	see paragraph 3.1
`--comparison`	if set, input markers will be appended in the output log file per line	—
`-e, --equal_effects`	if set, all effects will be assumed to be > 0 (betas) or > 1 (odds ratios)	see paragraph 3.3.2
`--trim`	may be used to 'trim' the marker content to a desired subcontent	see paragraph 2.1.2
`--skiplines`	used to require skipping lines within input files	see paragraph 2.1.3
`--stats`	will write out a little summary statistics file	—
`--version`	if set, YAMAS will report its version and revision number and exit	—

2.1.1.1 Threading

With the `-t`/`--threads` command line option the number of threads to use can be given. Per default only one thread will be used.

Using more threads a tremendous speed-up can be achieved. However, we recommend setting $1 \leq$ `threads` $\leq$ `ncpu`, where `ncpu` is the number of available processors on a shared memory computers or computing node. Setting `threads` above this number may increase performance, but carries the risk of slowing the entire system. Please respect the needs of other users on multi-user systems: If the system has may CPUs, still only some may be available at a given time.

Higher numbers of threads will essentially pay off with a larger amount of data and / or a more complex algorithm: For simple pointwise marker meta-analysis the calculation speed mainly depends on the hard disk speed and the CPU single core performance.

2.1.1.2 Algorithms

The `-a/--algo` command line option is used to select a particular algorithm to perform. Default is `pointwise`. The following table shows a full list of available algorithms.

Table 2.2: *List of available algorithm options.*

Option	Description	Reference
`pointwise`	a marker-by-marker conventional meta-analysis will be conducted (this is the default, when no algorithm is specified)	see paragraph 3.4
`ldblockwise`	the marker with the "best" P-value per LD-block per data set is included in the meta-analysis	see paragraph 3.5.2
`fillwithproxies`	if a marker is not found within a particular project YAMAS tries to identify the marker with the highest LD to the one we are looking for	see paragraph 3.5.1

A full explanation is given in chapter 3 on page 13.

2.1.1.3 Logfile

YAMAS will always write a log file, or better - dumps data to a log. With the `-l/--logfile` option the name of the log file can be altered. The default name depends on the algorithm you choose.

YAMAS will refuse to overwrite log files. This is a safety measure. In order to overcome this issue you, as a user, have to choose a different log file name or deleted the previous log file.

An example will look like:

```
$ yamas -l my_comprehensive_log_file_name
```

We recommend to give only the base of a file name – and no suffix, as YAMAS will choose the appropriate suffix for each file (several may be written out per run).

2.1.1.4 Chromosome

The `--chromosome`-option exists only in its long version. You can use it to restrict the analysis on only one chromosome. Valid Chromosomes are the numbers 1-26. YAMAS follows the numeric chromosome notation of plink (see `http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml`):

Table 2.3: *List of numeric chromosome codes.*

Code	Description	Number
1-22	human autosomal chromosomes 1-22	1-22
X	X chromosome	23
Y	Y chromosome	24
XY	peusdo-autosomal region of X	25
	mitochondrial DNA	26

So far YAMAS makes no difference between the chromosomes. However, this might change in future versions[1].

[1] This, of course, is not anticipated for the meta-analysis itself.

2.1.1.5 Being Verbose

Using the `-v`-flag you can turn YAMAS to be pretty verbose: YAMAS will report frequently about what is going on in the background and how the progress is.

If you experience trouble starting the YAMAS meta-analysis because YAMAS complains about your input file, you can also use the `-v`-flag. If set, the option will cause YAMAS to print the expected parameter type, the column you specified and the header column it can read.

In order to do so, YAMAS requires a header row starting with # in each data file:

```
# field1 field2 field3 ...
```

If your configuration file is set up like the one in the following paragraph (paragraph 2.1.4), the output will look like:

```
          Column - You specified : YAMAS read
      chromosome -              2 : field2
          marker -              1 : field1
        position -              3 : field3
...
```

The order of the output is a bit arbitrary[2].

2.1.2 Trimming

Sometimes only a handful markers should to be analysed. If YAMAS is invoked with the `--trim` option, all data sets are entirely read in, but output will only be calculated for the desired subcontent of markers. Unlike other options `--trim` may be used in three different ways:

- ```
  $ yamas --trim
  ```

  Invoked without further arguments the final marker content will be identical to all valid markers within the first project data set.

- ```
  $ yamas --trim rs123,rs1234,rs12345
  ```

  If supplied with a comma-separated list of marker IDs, only the given markers are in the final analysis. Note, that there is no space between those marker IDs, just commas. Parsing the command line would be tedious and error-prone, if spaces were permitted.

- ```
  $ yamas --trim filelist.txt
  ```

  Alternatively marker IDs may be supplied in a file. As always with YAMAS, gzipped files are allowed. The file has to contain one marker per line. Lines starting with a hash-mark ('#' as the first character) are considered comments.

2.1.3 Skipping lines

As mentioned in paragraph 2.1.4 lines starting with a hash mark (#) can be used to "outcomment" lines. If this requires the manual manipulation of several input files, you can use the `skiplines` option to ask YAMAS to ignore the requested number of lines, e. g.:

```
$ yamas --skiplines=4
```

will cause YAMAS to ignore the first 4 lines in *all* the input files. Currently it is not possible to skip a different number of lines in different input files.

[2] Lexically sorted due to programmers laziness.
```

2.1.4 The Configuration File

As YAMAS requires many input parameters, using very long command line options would be awkward. Instead, a configuration file is used. Per default YAMAS looks in the current working directory for a file called `yamas.cfg`. If not present the program aborts and tells you that it lacks input. However, with the command line options `-c` or `--config` you can define a different path and file name like this:

```
$ yamas -c path_to_your_configuration_file/your_choosen_file_name.cfg
```

Even the file suffix is completely arbitrary and up to you.

The file itself should be easy to understand. You can find a template in the `data`-directory of your YAMAS download. Here, we show you a template:

```
# Filenames containing the association data
# (your actual input data files)
assocfiles  = example_data1.csv example_data2.csv

# column positions of the marker ids
marker_cols = 1 2

# column positions of the chromosome ids
chromosome_cols = 2 1

# column positions of marker positions
position_cols = 3, None

# column positions of marker's effect allele
effect_allele_cols = 5,3

# column positions of marker's other allele
other_allele_cols = 6;4

# column positions of marker's effect (beta or odds-ratio)
effect_cols = 10 5

# column positions of marker's effect standard error
weight_cols = 11;6

# column position of marker's p-value
P-value_cols = 12,7
```

Let's walk through it step by step:

- Any line starting with a # is considered a comment line[3].
- All lines with `_cols` are variable declarations. Do not change the variable / parameter names. We hope their naming is obvious. However: You, as a user, are asked to specify the columns in your input data files for markers (rs numbers), chromosomes, position of the marker (in bp as in snpdb), of the effect allele, of the "other" allele, of the effect (as odds ratio or beta estimate or frequentist value), of the standard error (STDERR), and the corresponding P-value. Please look in the example directory of your download. You will find several example files, too.

[3] In fact, anything with # in front will be considered a 'comment' in a configuration file. However, in data files (see paragraph 2.2) only entire lines can be commented out with the #-mark at the first position of a line.

- First you need to name the input data files holding your association data. These can be Plink (**?**) or INTERSNP **?** output files, for instance.
- Subsequently all column numbers need to be given for each parameter. First for file 1, then file 2, then file 3 (if present) - and so on.
- As you can see, values can be separated by a single space, by commas, or a semicolon. You may mix theses separators at will.
- Also note the 'None' value for the second `position_cols`'s entry: All strings or numbers < 0 will be recognized as unset parameters. This is no problem as long as a specific parameter is not mandatory. **Mandatory parameters are: Marker id, chromosome number (1-26 are valid), effect, standard error, and the P-value.** Depending on the algorithm also the position number is mandatory (e. g. filling proxy alleles requires positions for non-arbitrary output). YAMAS will issure warnings or throw errors at you, if it cannot cope with its input files. Please read them carefully and report anything you do not understand.

YAMAS offers a way to automatically generate configuration files for some of the more common formats if all the input files are written in the same format:

```
$ yamas --writeFORMATconfigfile="infile01.extension infile02.extension \
infile03.extension ...."
```

The names of the files can be separated by any number of spaces, "," or ";".

Table 2.4: *List of available configuration file generator commands.* See notes below.

Command	Default input file extension	Remark
`--writePLINKconfigfile`	*.assoc	from standard '-assoc'
`--writeINTERSNPconfigfileLogistic`	*.txt	from logistic regression
`--writeINTERSNPconfigfileLinear`	*.txt	from linear regression
`--writeINTERSNPconfigfileLinear`	self-choosen	—
`--writeYAMASconfigfile`	.assoc	—

Currently, there is no way to automatically generate configuration files if not all of the input files are formatted in the same way. We strongly recommend to cross-check your input files with automatically generated configuration files.

2.2 Data Files

All input data has to be plain ASCII-text, ordered in columns. As column separator(s) may serve whitespace, commas and semi-colons - or any combination of these. The hash marker (#) serves as an indicator for a comment line, if placed as the very first character of a line. We recommend that you establish the first line of a file as comment or header line, e. g. like this:

```
# CHR         SNP      BP  EA  RA       OR      BETA       SE        P
    1   rs3094315  742429   C   T   0.8608   -0.1499   0.0240   0.2357
```

The first line of a file is interpreted as the line containing column identifiers. These are written to the shell for your convenience: This way configuration files can be checked at a glance (see paragraph 2.1.1.5 on YAMAS's verbosity).

Essentially all non-numeric input has to be escaped with the hash mark as the very first sign of a line. Else YAMAS will try to interpret that line as input data – which will fail and leads to an abortion of a YAMAS run. However, only the first line's content will be written out as described. The only exception to this rule is skipping lines as described in paragraph 2.1.3

All column identifiers in the comment line are completely arbitrary and only for your own purpose as a user.

All the columns in the example are mandatory – with the exception of the `OR` **or** `BETA` column, where only one will be used to determine the effect size (see below):

Table 2.5: *List of mandatory input data columns.* Columns labels in the header line are completely arbitrary and your choice.

Description	Explanation
the marker id	Should contain the SNP or CNV id as the comparisons require an universal identifier. Any unique string can be used as an id – of course, we thought of rs ids when we were designing the software.
the chromosome where a maker is located	To minimize memory usage YAMAS reads in data chromosome wise to ensure "supersets" of common data. The chromosome has to be a number in the range 1–26 (see paragraph 2.1.1.5).
the marker position on that chromosome	YAMAS offers to compare markers in LD blocks, genes, pathways, etc.. Lacking other criteria the assignment is based on a marker's position (see paragraph 3.5.2. The position will be interpreted as an integer number. Avoid using floating point input for this column: 1.25E3 can be anything around 1250, but is not precise!
the markers effect and reference or "other" alleles	These columns are used to issue control the input and to issue warnings, if mismatches occur. YAMAS will proceed analysing your data, but attempts to tell you, where pitfalls in later analysis or interpretation might be.
the effect size	The effect size can be anything. We expect you to use PLINK's `OR`, SNPTEST's `frequentist` output, INTERSNP's β or `OR`, or something similar. However, the effect should either follow a normal distribution with $\mu = 0$ or be convertible: YAMAS tries to convert `OR` to β-estimates ($\ln OR = \beta$), if asked for with the `--odds_ratio` command line option.
the standard error for this marker	This information is used to weight the effect sizes.
the p-value of the previous analysis	Not strictly necessary, this column is can be used to compare input in output p-values.

Let us take Plink (**?**) output as an example. The call

```
$ plink --file <somefile> --assoc --ci 0.95
```

will generate output like:

```
 CHR          SNP          BP    A1       F_A       F_U    A2         CHISQ
P            OR            SE          L95           U95
   1   rs12562034      758311     A    0.0907    0.1194     G         3.813
0.05085        0.7358        0.1575        0.5404         1.002
```

The `--ci` is necessary to produce the standard error column.

Here the configuration file would show those column number:

```
assocfiles  = your_plink_output.assoc
marker_cols = 2
chromosome_cols = 1
position_cols = 3
effect_allele_cols = 4
other_allele_cols = 6
effect_cols = 10
weight_cols = 11
P-value_cols = 9
```

2.2.1 Compressed Data Files

File sizes in GWAS studies can be enormous, hence compression sometimes is a must. In order to avoid the cumbersome compression-uncompression steps YAMAS is able to read gzipped files on the fly. This file format is common on UNIX environments, but also provided for Windows (`http://www.gzip.org/#exe`). The files have to have the `.gz`-suffix in order to be recognized by YAMAS. Recognition is automatic, no further arguments have to be given by you, the user.

2.2.2 Auxilliary Data

2.2.2.1 Proxy Markers

The algorithm which assigns proxy markers in case of missing markers in one study panel (see paragraph 3.5.1 for details) requires a particular reference format. One such file is provided for download (see `http://yamas.meb.uni-bonn.de/`). It is also possible to generate such files with INTERSNP (**??**; see paragraph 3.5.1.2). In any case the file has do adhere the following format specified in paragraph 3.5.1.2.

2.2.3 Output files

YAMAS will generate output files ending on `.log`. Output files will look like this:

Table 2.6: *List of column header and their meaning.*

column name	content
`STATUS`	status line for the maker, usually just OK, but might contain warnings
`CHR`	chromosome of that marker
`SNP`	id of the SNP or marker
`BP`	base pair position of that SNP
`EA`	effect allele, which is chosen to be the minor one
`OA`	other or major allele
`DIRECTIONS`	the directions of the effects per project, see below
`EFFECT`	the averaged, weighted effect – either as odds ratio or β-estimate
`WEIGHT`	the averaged, weighted weight – usually the standard error
`CI`	confidence interval for the fixed effect with default confidence level 0.95
`P`	the P-value
`Q`	P-value for between study variance, see paragraph 3.2
`I^2`	the I^2-statistics for the between study variance, see **?**
`EFFECT(R)`	the random effect, see paragraph 3.2
`WEIGHT(R)`	the random effect weight, see paragraph 3.2
`CI(R)`	confidence interval for the random effect with default confidence level 0.95
`P(R)`	the random effect P-value, see paragraph 3.2

Some explanations:

- The status line might contain warnings (see paragraph 2.2.3.1 about position or chromosome mismatches between markers or – in the case of the "proxy algorithm" inform you that a proxy marker has been entered.

- The directions inform about the effect direction of the analysis per project:

Table 2.7: *"Directions" and their meaning.*

direction	meaning
`+`	the direction of that marker was positive (odds ratio > 1 or β-estimate > 0)
`-`	the direction of that marker was negative (odds ratio ≤ 1 or β-estimate ≤ 0)
`x`	the marker of that project was present, but invalid
`?`	the marker was missing for the respective project.
Example:	
`--+x-?`	Here, the marker for the first, second and fifth project had a neg. direction. The direction was positive within the third project and not valid in the fourth. The last (sixth) project did not show that marker at all.

What, if the markers alleles will not match those of the proxy (see paragraph 3.5.1)? `yamas` checks for HAPMAP/1k annotation conformity and will, if necessary work with the complement alleles – no warning will be issued, but you can recognize this if the alleles in the final output won't match the input.

Here is how the logfile columns are extended in case of the proxy-algorithm (see paragraph 3.5.1) if you choose to add the `--comparison` flag (see paragraph 2.1). It then contains the following columns in addition:

Table 2.8: *List of column header and their meaning. These values are only given, if the marker was a proxy marker and the proxy-algorithm (see paragraph 3.5.1) was chosen.*

column name	content
`R^2`	the R^2 for a proxy marker or 0 if the marker was no proxy marker
`PROXYALLELE`	the proxy allele for a proxy marker or `None` if the marker was no proxy marker
`STRANDSWITCH`	0 or 1, indicating a absence or presence of a strand switch for a proxy marker or `None` if the marker was no proxy marker

2.2.3.1 Status Messages Issued by YAMAS

YAMAS tries to be as tolerant as possible: Instead of aborting runs, the program will issue warnings in the status column. We know that you will probably parse the YAMAS output for further processing (e. g. in a Manhattan plot). Hence, all words in a status report are concatenated by the tilde character ('∼').

Table 2.9: *Status messages and their meaning.* The table reflects warning precedence is the one in the table. No 'position'-warning will not be given, when the proxy algorithm is called.

message	meaning
`OK`	alas, YAMAS did find nothing to complain about
`WARNING:~chr.~mismatch`	indicator for a serious flaw in the datasets: markers with the same ID are reported to located at different chromosomes
`WARNING:~1~OR~was~zero`	at least 1 given odds ratio was zero. This raises a problem for YAMAS, see paragraph 3.2.1 for further information.
`WARNING:~position~mismatch`	the marker was missing for the respective project.
`INFO:~proxy~with~r^2~=1`	a proxy marker with r^2=1 was added (only applicable when the appropriate algorithm was called, see paragraph 3.5.1)
`INFO:~entered~proxy`	a proxy marker was entered (only applicable when the appropriate algorithm was called, see paragraph 3.5.1).

Once one message other than `OK` was issued, no other warning or info is issued – although more than on warning might apply.

Chapter 3

Algorithms in YAMAS

3.1 Performing Association Tests

Right! – Association testing is not part of meta-analysis itself. We simply mimicked the PLINK (**?**) association testing routine.

YAMAS does not intend to compete with designated testing software such as PLINK (?), INTERSNP (?), or SNPTEST (?). For most cases, we recommend using one of those programs as they are specialized software tools which most certainly provide the desired statistical test, while YAMAS's association testing is most simple. Particularly YAMAS is not capable of dealing with quantitative traits while other tools are.

That being said, we still thought that implementing this feature has its use: If meta-analysis should be done within the schedule of a consortium's work plan, every partner has to provide files with association data in a designated format. This can be hard – to say the least – as it is always hard to coordinate many partners. In this case YAMAS might come to the rescue: Simply let every partner perform the basic association test (if our test is actually applicable) to their data and compare the results with YAMAS.

YAMAS works on a standard `tped/tfam` file tuple (see `plink`, `http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml`, for details). You may invoke YAMAS like this:

```
$ yamas --assocfilename=path_and_name_of_your_tped_file_to_be_tested \
        --missing=[missing character]
```

If the `assocfilename` is given no meta-analysis, but a simple association test (see below) is done. The `missing` flag can be used to specify the missing character in your `tped` file. It defaults to `0` and is optional.

After the run a file ending on `.assoc` is created. It contains the following columns:

Table 3.1: *List of column header and their meaning.*

column name	content
CHR	chromosome of that marker
SNP	id of the SNP or marker
BP	base pair position of that SNP
EA	effect allele, which is chosen to be the minor one
F_CA	frequency of this allele for cases
F_CO	frequency of this allele for controls
OA	other or major allele
OR	odds ratio for EA
BETA	beta estimate (simply $\ln OR$)
CHISQ	basic allelic test chi-square (1df)
SE	the standard error
P	the p-value of the χ^2 test
P_ARMITAGE	the p-value of the Armitage Trend test

3.2 Math

All math of a meta-analysis is fairly standard, a good overview for the case of GWAS-data is provided by **?**. In this text we will name our effect sizes E and the weight of an effect w.

3.2.1 Preliminaries

YAMAS accepts odds-ratios ($\mathrm{OR}'\mathrm{s}$) given as the effect, yet to perform our meta-analysis we need the effect sizes to be distributed around 0, therefore we assume

$$E = \ln(\mathrm{OR}) \,. \tag{3.1}$$

If YAMAS is asked to work with odds-ratios all later output for the effect size will be odds-ratios again.

What if $\mathrm{OR} = 0$? $\ln(\mathrm{OR})$ would be $-\infty$, right? Therefore, we came up with an *ad hoc* solution and add constant value of 0.001 to such odds ratios. Hence, the final result should be read with care. Do you know a better solution? Please let us know. At least, we issue a warning message in such cases, see paragraph 2.2.3.1.

Likewise, usually testing output gives the standard error of an test, yet for meta-analysis we need the weight for a particular effect size. Therefore we take

$$w = \frac{1}{\text{standard error}^2} \tag{3.2}$$

3.2.2 Fixed effects

With equation 3.1 we are able to calculate the weighted mean for both, odds-ratios and beta-estimates:

$$\bar{E} = \frac{\sum_{i=1}^{k} w_i \cdot E_i}{\sum_{i=1}^{k} w_i} \tag{3.3}$$

and our averaged standard error is

$$SE_{\bar{E}} = \sqrt{\frac{1}{\sum_{i=1}^{k} w_i}} \,, \tag{3.4}$$

for every dataset i and k datasets.

We finally compute a z-value

$$z = \frac{\bar{E}}{SE_{\bar{E}}} \ , \tag{3.5}$$

which we can use to calculate a p-value with a two-tailed test:

$$p = 2 \cdot (1 - \Theta(|z|)) \ , \tag{3.6}$$

where $\Theta(|z|)$ is the standard normal cumulative density distribution function.

3.2.3 Random effects

The above fixed effects model holds, if we assume that all studies essentially share a "true" effect. This assumption may be implausible, when comparing GWASes: Population stratification may imply different "true" effects for different markers. It may also happen that different studies in your sample considered covariates in a different way, e. g. studies working with different control cohorts, some may be general "randomly" selected individual sample and others may rely on controls from selected non-trait individuals. In such cases we should allow for a distribution of "true" effects.

For this we need to decompose the variance. We compute the total variance (the observed one) and isolate the within-study variance. The difference between these values gives us the variance between studies, which we call τ^2 (**?**). We may consider three cases:

a) The "ideal" (and most reliable) case is that all study effects are close to their mean and each study shows little variance. Therefore, we are inclined to believe: There is no variance between the studies – τ^2 is low (or zero).

b) There is variance between studies, but each study's variance is high and this explains the between-study variance: Given the imprecision of all studies we expect the studies to vary from one to another. Again, τ^2 is low (or zero).

c) If there is variance between studies and each study's variance itself is low (which is frequently the case for quality-filtered GWASes!), we cannot explain the between-study variance and the excess variation will be reflected in a higher τ^2.

Obviously τ^2 increases if the variance between-studies decreases and/or if the observed within-study variance increases.

YAMAS calculates a Q statistic (originally proposed by W. Cochran (**?**); see **?** for a nice overview), which represents the total variance[1]:

$$Q = \sum_{i=1}^{k} w_i \left(E_i - \bar{E}\right)^2 \ . \tag{3.7}$$

If the only source of variance would be the within-study error, the expected value of Q would be the degrees of freedom (df) of the meta-analysis. This consideration allows us to compute the between-study variance:

$$\tau^2 = \begin{cases} \dfrac{Q - df}{\left(\frac{\sum_{i=1}^{k} w_i}{\sum_{i=1}^{k} w_i E_i^2}\right)} & \text{if } Q - df > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The numerator $(Q - df)$ is the excess variance, the denumerator is a scaling factor, which we need because Q is a weighted sum of squares. With the denumerator, we ensure that τ^2 has the right scale[2].

[1] Internally YAMAS uses an extended version of the equation to compute Q.

[2] Note that due to equation 3.8 τ^2 is actutally a biased estimator.

With τ^2 we are enabled to change equation 3.2 to

$$w^* = \frac{1}{\left(\text{standard error}^2 + \tau^2\right)} \ . \tag{3.9}$$

All equations for the fixed effect model stay unchanged, except the replacement of w_i to w_i^*.

What did we learn so far about the random effects model? Without true heterogeneity among for the effect estimates the between-study variance (τ^2) is zero, in which case random and fixed effect model are identical.

YAMAS however does not include the Q statistic in its output but the p-value for this statistic. Q is χ^2 distributed with df degrees of freedom (**?**). Hence the p-value is analogous to equation 3.6:

$$p = 2 \cdot (1 - X(Q)) \ , \tag{3.10}$$

assuming X to be the χ^2 cummulative density distribution function.

3.3 Treating effects

3.3.1 Effect Allele vs. "Other" Allele

YAMAS talks of effect alleles and other alleles. This might be confusing at first glance, yet as the community uses a variety of names designating the allele(s) where tests are conducted for, YAMAS chooses to use these names as identifiers which cannot be mistaken.

A problem in GWAS meta-analysis is that partners within a consortium all need to agree upon allele designations: It is not uncommon to perform a test on the minor allele. But what actually is a minor allele? The one (for a particular marker) in the entire data set across all partners within that consortium? The one in a sub data set? The one in the cases of that sub data set? The one in the controls, only? You get the point.

Now, in order to avoid such mismatches YAMAS refers to effect vs. other allele. Here are the definitions:

A **effect allele** is the allele, where the performed test refers to. A **other allele** is the other one.

We strongly recommend to tabulate both alleles under all circumstances. This ensures that the one actually performing the meta-analysis will be able to find apparent mismatches.

To lift the lid of the black box "YAMAS", imagine the following case:

Project Site	Effect	EA	OA	direction output
Site 1	0.3	G	T	+
Site 2	-0.2	T	G	+

The result of project site 2 will be multiplied by -1 as the effect and other alleles are exactly switched and it can be assumed that the effects actually share the direction.

3.3.2 Equalizing all effects

The above explained switching of effect directions may not be desired, for instance if your sample do not reliably report the effect alleles or for cross-disorder comparisons. In this case YAMAS can be asked to neglect the sign of each effect and treat it *always* as positive using the `-e` or `--equal_effects` switch.

3.4 Point-Wise Marker Comparison

Using YAMAS standard settings – without additional algorithm selection – will cause YAMAS to perform a marker-by-marker meta-analysis: Every marker is compared with its counterpart in the other projects.


As for YAMAS the marker's base pair position is needed to attribute proxy markers when using the "proxy algorithm" (see paragraph 3.5.1). Hence, it is of prime importance that markers with identical id do share identical position values. Performing a marker-by-marker meta-analysis lets YAMAS control your input. A warning will be written in log file for every marker where the positions across different cohorts do not match.

3.5 Using Reference Panels

With incomplete marker panels (e. g. because not all sample were imputed) YAMAS is able to fall back to reference panel information. The following two sections describes two approaches which we implemented.

3.5.1 Inserting mutual Proxies identified by Linkage Disequilibrium

As mentioned, meta-analysis usually requires to bring all involved marker panels of different studies to (more or less) the same content. One way to go is to impute all data sets to the (common) content of the HapMap- (**???**) and / or the 1000-genomes-project (**?**). The alternative is to accept a substantial power loss of the meta-analysis by analysing markers, SNP by SNP, for the common content per marker.

YAMAS, now, is capable to avoid this power loss without the need to impute the data: By using reference data from the HapMap and 1000 genomes projects users are enabled to analyse all SNPs that are present in at least one of the experimental marker panels: LD-information is used to find substitute makers for those missing. For each SNP that is missing in one study, the marker from the study with largest r^2, according to HapMap data, with the missing marker is chosen as a "proxy SNP". Furthermore, based on HapMap haplotype frequencies, "proxy alleles" of a SNP and its proxy-SNP are identified, i.e. it is decided which allele of the missing SNP predominantly occurs in combination with which allele of the proxy-SNP. For each SNP present in at least one study, MA of that SNP is done by combining the association results of the SNP or, if not available the proxy-SNP, across studies. Consistency of the direction of effects is automatically accounted for through the proxy allele definition.

Using conventional MA and / or the LD-based information gathering method YAMAS offers an easy approach to MA of GWAS studies and access to any tabulated format of (GW)AS results. Suitable input reference files are available at the YAMAS download page.

3.5.1.1 Download and Usage of Reference Panels

All reference file can be found on the yamas web page: `http://yamas.meb.uni-bonn.de/`). Please download the appropriate file and save it to a location YAMAS is able to access. The current (as of version 0.8) source for the YAMAS reference panel is a mix of Hapmap3, build 36, data and 1000 genomes, pilot 1, data (see `ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/release/2010_03/pilot1/CEU.SRP000031.2010_03.genotypes.vcf.gz`).


The file size of a reference panel can be enormous: In case your download failes or your connection is to slow, please contact us – me might find a solution for you.

In order to find the reference file it needs to be specified by an additional keyword in the configuration file, e. g.:

```
# file with LD-proxy information
Proxyfile = <path>/proxyreference.txt.gz
```

Also, YAMAS needs to be asked to perform this algorithm by using the `-a`/`--algo`-option:

```
$ yamas --algo fillwithproxies
```

Per default all mutual markers with an $r^2 \geq 0.2$ are considered. In order to change this setting, use the `--r2threshold` flag:

```
$ yamas --algo fillwithproxies --r2threshold 0.4
```

This will change the threshold to 0.4.

In any case the default output file suffix is `yamas_proxyanalysis`. This logfile contains additional columns. They are listed and explained in table 2.8.

3.5.1.2 Producing Own Reference Files

With INTERSNP (**??**) (see `http://intersnp.meb.uni-bonn.de/`) it is possible to produce a file with r^2-values for all SNPs that lie within a pre-specified distance, for instance from HAPMAP and/or 1000 Genomes data (tped/tfam files). In addition to the basic keywords (TPED/TFAM, qc options,..) of `intersnp` (see the INTERSNP-manual), the following keywords have to be specified in the INTERSNP-selectionfile:

Table 3.2: *List of additional* INTERSNP*-keywords* Those keywords have to be specified in the INTERSNP-selectionfile in order to produce a list of putative mutual proxy markers.

Additional keyword	Setting
TWO_MARKER	1
TEST	13
QT	1
HAPLO	1
HAPLO_DIST	10000
DOHAPFILE	1

In the above example, r^2 will be computed for all SNPs with a distance smaller than 10000 bp. The outputfile (*hapfile.txt) will contain all SNP pairs for which r^2>0. An excerpt might look like:

```
1 rs2462492 T C rs12752391 A G 955147 0.113 0
1 rs2462492 T C rs34820586 C G 992041 0.131 0
1 rs2462492 T C rs34381538 A G 1006926 0.132 0
```

Here, the columns have to be read as follows:

Column 1) chromosome

Column 2) marker id of the first marker, e. g. rs-id of the first SNP

Column 3) minor allele of the first marker or SNP

Column 4) major allele of the first marker or SNP

Column 5) marker id of the second marker, e. g. rs-id of the second SNP

Column 6) minor allele of the second marker or SNP

Column 7) major allele of the second marker or SNP

Column 8) the difference of the markers base pair positions

Column 9) r^2

Column 10) identifies proxy alleles. In line 1, `0` indicates that the haplotype A-A (composed of columns 3 and 6, for each SNP the first listed allele) is more frequent than expected under linkage disequilibrium (LE). It immediately follows that also the C-G haplotype is more frequent than expected under linkage disequilibrium. Therefore, allele A from SNP 1 and allele A from SNP 2 are mutual proxy alleles (see paragraph 3.5.1), and likewise allele C from SNP 1 and allele G from SNP 2. In line 2, the situation is reverse, as indicated by a `1` in the last column. Here, allele A (column 3) from SNP1 and allele T (second(!) allele of SNP 2, column 7) are the mutual proxy alleles.

A more detaile output is can be generated with the choice DOHAPFILE 2 when invoking INTERSNP, e. g.[3]:

```
16 rs1211375 A C rs3918352 A G 0.378 0.622 0.425 0.575 0.300 0.078 0.125 0.497 0.641 0.338 0
16 rs1203957 T G rs3918352 A G 0.110 0.890 0.425 0.575 0.110 0.000 0.315 0.575 1.000 0.167 0
16 rs3918352 A G rs11642609 C T 0.412 0.588 0.485 0.515 0.000 0.412 0.485 0.103 1.000 0.659 1
```

The first 7 columns are as before. Columns 8-9 are allele frequnecies for SNP1, columns 10-11 allele frequencies for SNP 2. Columns 12-15 are the 4 haplotype frequnecies, here in lexicographic order, i.e. ignoring the proxy definition. Column 16 is D', column 17 r^2, and column 18 is the proxy indicator. The computation of the indicator is based on the allele and haplotype frequencies listed.

3.5.1.3 The Algorithm Itself

This paragraph shows the algorithm in pseudocode. Comments are preceeded with a ▷ sign.

First YAMAS needs to read the mutual proxies:

3.5.1.4 Problems

In case a proxy marker with r^2-value equal to 1.0 is entered, the case is ambiguous: Such r^2-values are only possible, is such one marker was monozygous. If such a proxy marker was used in the meta-analysis, information about it is written in the status column (see paragraph 2.2.3.1). We advise to examine interesting results further with `intersnp`'s haplotype analysis algorithm (see `http://intersnp.meb.uni-bonn.de/`).

3.5.2 LD-blockwise Analysis

Chapter remains to be written.

[3] Sorry for the rather tiny font size of this example, but else it would not fit on the page.

Appendix A

Utilities

This chapter attempts to describe all utilities easing your work with YAMAS. Note, that this documentation might not describe all you as a user might need to know: Just inform the YAMAS-developers about any additional information you require. Also, please inform us if you would like to provide a script which eases the work with YAMAS or would like to write us additional utility scripts.

All YAMAS Python scripts have a shebang and a `__future__`-statement:

```
#! /usr/bin/env python
from __future__ import with_statement
```

The shebang (`#! ...`) lets unixoid operating systems find a script in the local `PATH`, if made executable. The `__future__`-statement limits the compatibility to $2.6 \leq \text{Python} - \text{version} < 3$.

A.1 Forest Plots

A forest plot (sometimes called a "blobbogram") is a plot designed to illustrate the relative strength of effects in multiple quantitative scientific studies addressing the same question. In the case of GWAS-meta analyses a forest plot can be used to display the meta analysis result of a particular marker. Along with a YAMAS download a script which can produce a forest plot for a particular marker. The script is written in Python (`www.python.org`) and requires numpy (**?**) and matplotlib (**?**). Note that Python scripts are not compiled and require a Python interpreter in addition to the mentioned packages. Please consult your packaging system or download the interpreter from (www.python.org).

Figure A.1 on the following page shows a typical forest plot generated by the `forest_plot`. The script is invoked like (UNIX or cygwin shell):

```
$ <path>/forest_plot --configfile <configuration file> \
                     --result <result or logfile> \
                     --rsid <marker id of the marker to be extracted>
                     --xlabel [label of the x-axis]
```

The command can be typed on one line, the \ omitted. The configuration file and the result file are the respective files for the analysis you as the YAMAS-user have performed.

In addition the `--dpi`-option lets you specify the resolution of the plot and the `--extension` the kind of the plot. Valid extensions are `png`, `ps`, and `eps`. Additional kinds may be available, depending on your matplotlib installation. Please consult the matplotlib manual.

The `--xlabel`-options lets you specify the label of the x-axis. LaTeXexpressions are valid, e. g. `--xlabel "$\\beta_i$"`. Note however, that backslashes have to be escaped. This is why we see the two \\ in the example.

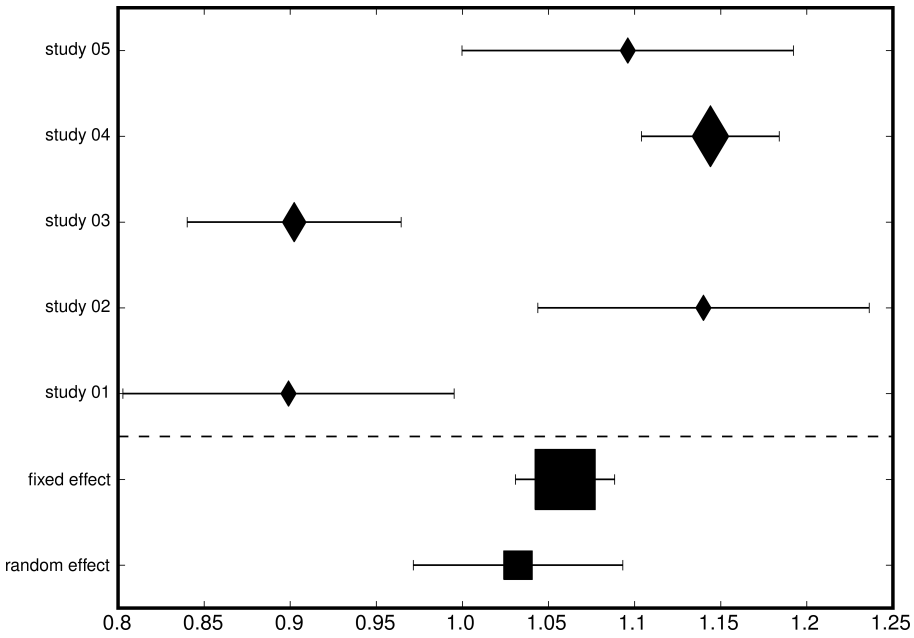

Figure A.1: Example forest plot for one particular marker showing how the more conservative random effect reflects the variance between studies. Note that the marker size (marker of the plot, not the GWAS) is inverse proportional to the standard error of that marker within a study or the meta-analysis.

A.2 Switching Genotypes

Occasionally studies may differ in there designation of effect and reference or other alleles for many markers. In this case YAMAS will refuse to consider these markers as valid, because the meta-analysis requires reliable effect directions and automatic corrections (described in paragraph 3.3.1) cannot be accomplished if the genotypes aren't the same.

In this case the `swith_markers`-script can change the genotypes of particular markers according to a reference. The script is invoked like

```
$ <path>/switch_markers --configfile    <configuration file> \
                         --referencefile <the study file which \
                        genotypes should serve as a reference> \
                         --wrongfile     <the study file which \
                        genotypes should be corrected>
```

The command can be typed on one line, the \ omitted.
Currently (as of YAMAS version 0.5) two cases are considered:

- The genotypes are identical - hence, the genotypes in `wrongfile` are left untouched.
- The genotypes are each others complements - here, the genotypes in `wrongfile` for the respective marker are changed to their complements.

A.3 Transforming `.vcf`-Files to Plink Compatible Format

While YAMAS provides refrence files for the "proxy-algorithm" (see paragraph 3.5.1, own reference files can be generated (see paragraph 3.5.1.2). For this purpose Plink (**?**) compatible files are needed. Some projects, e. g. the 1000 genomes project, provide their data in the `vcf`-format. To transform `.vcf`-files into Plink `tped/tfam` file tuples, please have a look at the `vcf2tped` script in the utility directory of your download.

A.4 More Utilities?

Obviously we cannot provide solutions for all imaginable problems which may come with meta-analysis. If you encounter a so far unsolved problem, please contact us: Either provide a general solution for that problem as a script (not necessarily Python) – your name will be mentioned! – or ask us for help, such that we can provide a script to solve your problem (in more complex cases we might ask for co-authorship).

Appendix B

Shorties

This chapter is a loose, non-comprehensive collection of useful tools for `yamas`-users.

Sorting files numerically the UNIX way:

```
$ sort -nk <column> file > sorted file
```

There is no need to unzip those really big reference files, used in `yamas`, as unzipping works on the fly without writing on the disk the unzipped file:

```
$ gunzip -c <path to gzipped file> | <command, e.g. grep>
```

Appendix C

YAMAS – Development

The following chapters are intended for developers of YAMAS. You are most welcome to participate! Contributions may not only be code-based: Any bug report, hint on how to improve the manual, or feature request is welcome.

C.1 How to retrieve a working copy of YAMAS

YAMAS is not anonymous! Therefore you need an account for our repository. Every contributing developer is welcome, please do not hesitate contacting us for a login.

A checkout requires SUBVERSION (see `subversion.tigris.org`). This versioning system is available for (almost) every operating system. Please contact your local administrator, if you don't know how the program can be installed.

A first checkout can be done as follows:

```
svn co --username <username> \
 svn+ssh://<username>@131.220.23.67/daten/svnrep/yamas <path>
```

You can ommit the \-sign and type everything in one line. Everything between `< ... >` should be replace by the appropriate username and path. `path` can be any path you like, including the current working directory (`.`).

Subsequently only use `svn update` after you did the first checkout. Subversion checkouts should be performed once and only once per working directory!

C.2 Building the development version

YAMAS uses `scons` (see `www.scons.org`) as a build system (UNIX's `make` is the most well-known counterpart). In order to build a svn download just type

```
$ cd your-yamas-download-path
$ scons
$ scons -c # will clean up after building
```

`scons` is available for download on the `scons` web page. However, many systems provide packages for `scons`. Please consult your sys-admin if in doubt.

C.3 Colorized Compiler Output

The YAMAS `SConstruct`-file prepares for using `colorgcc` (see `http://schlueters.de/colorgcc.html`). Using `colorgcc` is optional, if a colorized compiler output is appreciated you will have to install `colorgcc` locally in your `$PATH`.

C.4 Packing a Release Version


Update the `yamas` version in the `SConstruct`-file, first. `yamas` follows the GNU version numbering scheme: major.minor.revision .

Prior to packing a release, please update the repository first:

```
$ svn update
```

This is because `yamas` can report is version and revision number and updating first makes sure that users reporting bugs can be sure about the release.

Typing

```
$ scons --static
# and subsequently !!!
$ scons --pack_release
```

will create a directory called `dist` with the appropriately packed files for the current system.

Right now, there is a variable called `file_list` in the `SConstruct`-file which describes the files to be packed, like this:

```
file_list = [('../yamas', './yamas_%s/yamas' % VERSION),
             ('../LICENSE', './yamas_%s/LICENSE' % VERSION),
             ('../trunk/data/yamas.cfg',
                 './yamas_%s/examples/yamas.cfg' % VERSION),
             ('../trunk/doc/Manual.pdf',
                 './yamas_%s/doc/Manual.pdf' % VERSION)]
```

Read this line as follows: The first item or string in each line is the file to be packed, the second the path where it should end after unpacking/-taring/-zipping the packed files. Please feel free to add all files, which might be useful for potential users after download.

The `VERSION` parameter is just the version string, which will be used for naming the download directory as a unique identifier of the YAMAS version.

A file containing the md5 sums for all produced files is written out, too.

C.4.1 Supplying the Download Server with Large Files

Our download server is capable of handling pretty big files, yet `yamas` reference files can be too be for the CMS. So, please copy any big file directly to: `/srv/www/htdocs/yamas/dm_documents` at 131.220.23.75. An account can be created upon request.

C.5 Building a Debug Version

Invoking SCONS like

```
$ scons --debug_build
```

Will compile and link with `-g`, `-pg`, and `-DDEBUG` flags. The first two flags enable using the GNU debugger (by writing a file named `gmon.out` when YAMAS runs). The third flag turns internal error handling routines of YAMAS off.

C.6 Selecting Compilers

For an explicit compiler seletion try:

```
$ scons --compiler=g++-4.4
```

for instance.

C.7 Parallel Builds with Scons

Like the infamous GNU `make` tool, SCONS allows using the `-j` option to build an application in parallel:

```
$ scons -j <number of parallel compiler calls>
```

C.8 Including Sandboxed Code

If a Developer wants to create sandbox code, that is code which should not turn up in productive versions of the program until completion of that code, the code in question can be framed with preprocessor guards like this:

```
#ifdef SANDBOX
// your sandbox code here
#endif
```

This guard can be applied to entire `cpp`-files.

Issuing the `--sandbox`-flag to scons will turn the `SANDBOX`-preprocessor variable on.

C.9 Other Options

```
$ scons --help
```

will list a bundle of additional options we included to ease the YAMAS development.

C.10 Used Libraries

C.10.1 OpenMP

YAMAS uses `OpenMP` to spawn threads on shared memory systems. This way some algorithms can reach a tremendous speed-up. Please install the appropriate header for your system using the system's packaging tools.

`gcc` from version 4.4 on will support `OpenMP` 3.0. We recommend using gcc as the compiler. However, necessary instructions for other compilers are linked here: `http://openmp.org/wp/openmp-compilers/`.

C.10.2 boost

YAMAS makes heavy use of `boost` (see `http://www.boost.org/` for download and instructions). In order to ease the installation process, here a shortcut for Unix-like systems:

```
$ # need to install MPI library
$ # Debian and Ubuntu users:
$ sudo apt-get install lam4-dev libopenmpi-dev
```

Or subversion source code checkout:

```
svn co http://svn.boost.org/svn/boost/trunk boost-trunk
```

boost checkout is anonymous. See `http://www.boost.org/users/download/`. Subsequently only use `svn update` after you did the checkout. Subversion checkouts should be performed once and only once per working directory!

Installation:

```
$ cd your-boost-download-directory
$
$ ./bootstrap.sh --help # shows configuration options
$ ./bootstrap.sh
$ ./bjam # this will take a while
$ # and finally
$ sudo ./bjam --with-regex install
```

If you don't have administrator rights:

```
$ ./bootstrap.sh --prefix=path/to/installation/prefix
$ ./bjam
$ ./bjam --with-regex install
```

The `--with-regex`-flag is important!

Troubleshooting:

- YAMAS does not link against `boost::iostreams`? One reason might be that this particular library did not compile. In this case try setting the environment variable `NO_BZIP2` to 1 and re-run `bjam`. For bash-like shell the command is "`export NO_BZIP2=1`".
- YAMAS does not link at all? Please set the enviroment variable `LIBRARY_PATH` to the appropriate path. Consult your local administrator if you don't know how.

C.10.3 zlib

YAMAS uses `boost::iostreams` to decompress zipped files on the fly. In order to compile `zlib`-headers are required. See `www.zlib.net` for detail and / or consult your systems packaging system.

C.11 Programming Standards

The purpose of these standards is to provide a consistent, professional look for the source code. Deviations from the general rules below are acceptable if they enhance the readability of the code.

That being said, it's perfectly ok, if you do not adhere to the following standards – as long as the code stays understandable.

- Tabs: There should be no embedded tab characters in the source code.
- Indentation: Normal indentation should be 4 spaces.
- Braces should start directly after an opening statement – separated with a space[1]. Example:

```
for (i = 1; i = 10; i++) {
   if (test) {
      block of code
   }
}
```

- Braces should start one line below an opening statement[2]. Example:

[1] This is the preferred style of one developer (C. Meesters).
[2] This is the preferred style of a second developer (T. Becker).

```
for (i = 1; i = 10; i++)
{
   if (test) {
      block of code
   }
}
```

- Pragmas may follow yet a different bracing style:

```
#pragma omp critical
{
some_vector.push_back(item)
}
```

  Justification: pragmas are a special compiler directive and – if braces are needed usually followed by short code segments, which should be easily “spottable” / distinguished by the reader.

- Braces are not required for one line conditions or very simple for loops[3].

```
if (test) return;
```

- Variable declarations:

  1. Each variable declaration should be in a separate line.
  2. Pointers and references should be associated with the variable name, not the type. That is, use the K&R and not the Strostrup style.
  3. If reasonable, line up variable names and any equal signs for assignments.
  4. Group variable definitions of the same type together.

```
int     x;
int     range;
string* lbl_a;
double  temperature = 72.0;
double  y           = 1.2;
```

- Class variables. Try to minimize these. When necessary, make public only when truly needed.

- Spacing:

  1. There should not be a space between a function name and the opening parenthesis.
  2. There should be a space after a programming construct (if, for, while, etc.)
  3. There should be **no** space after an opening parenthesis or bracket and before a closing parenthesis or bracket.
  4. There should be a space around operators such as !, =, <=, etc.

```
for (i = 0; i < 10; i++) {
     x[i] = function(a, b, c);
 }
```

- Try to keep line lengths to 80 characters or less. Occasionally exceeding 80 characters is OK if a line break would detract from the overall readability.

[3] Also disliked by one of the developers.

- If arguments to a function don't fit nicely on one line, split it like this:

```
int lots_of_args(int an_integer, long a_long, short a_short,
                 double a_double, float a_float )
```

- Line things up when it makes sense.