```
#####################################################################
# Boysen, Heuer, Tetens, Reinhardt and Thaller: A novel approach    #
# based on derived genotype probabilities in large cohorts of cows  #
# reveals dominance effects for milk production traits in German    #
# Holstein dairy cattle.                                            #
#####################################################################
#
#############################################################################
#  A.) R-scripts used for the simulations and regressions with simulated data   #
#############################################################################
#
# ------------------------------------------------------------------
# |   1. Single-marker scenario with completely unrelated cows.      |
# ------------------------------------------------------------------
# Functions:
# gt_estimate(n_cow, p_maf)
#   Creates and exports the matrix "gt_matrix" with genotype
#   probabilities and "true" genotypes.
#   Given are the number of cows and the minor allele frequency.
#    call: gt_estimate(200000,0.3)
#
# analyze(h2,domratio,a)
#   Based on the data in gt_matrix phenotypes are assigned to the
#   cows given the "true" genotypes of the cows.
#   Given are the heritability, the degree of dominance and the
#   additive effect in multiples of the genetic standard deviation.
#   Creates objects summary_reg (results of fit with genotype
#   probabilities), contrast_add and contrast_dom (results based
#   on true genotypes).
#   Requires libraries car and multcomp.
#    call: analyze(0.3,0.5,0.5)
#
gt_estimate <- function (n_cow, p_maf)
{
  #
  # simulation of genotypes:
  # - The sire (col1) and grandsire (col2) genotypes are
  #   sampled given MAF and Hardy-Weinberg equilibrium.
  # - The cows genotype probabilities are calculated based on
  #   MAF and sire's and grandsire's genotypes.
  # - The cows 'true' genotype (col3) is sampled given the
  #   deduced genotype probabilities.
  # - The regression coefficients stored in gt_matrix
  #   - additive regression coefficient (col4)
  #   - dominance regression coefficient (col5)
  #
  # Genotypes are coded 2,1,0 representing e.g. AA / AC / CC
  # Object as explained above
  gt_matrix   <- array(0,dim=c(n_cow,5))

  # probabilities for the three genotypes given Hardy-Weinberg equilibrium
  p_HWE_vec <- c(p_maf**2,2*p_maf*(1-p_maf),(1-p_maf)**2)
  #
  # simulate genotypes of sire and sire_dam (from the H-W population)
  #
  gt_matrix[,1] <- sample(c(2,1,0),n_cow,replace=T,prob=p_HWE_vec)
  gt_matrix[,2] <- sample(c(2,1,0),n_cow,replace=T,prob=p_HWE_vec)

  # calculation of genotype probabilities and regression coefficients,
  # sampling the 'true' genotypes of the cows.
  #
  for (i in 1:n_cow)
   {
    gt_sire        <-  gt_matrix[i,1]
    gt_siredam     <-  gt_matrix[i,2]
    p_QQ           <- 0.5*gt_sire*(0.25*gt_siredam+0.5*p_maf)
    p_qq           <- 0.5*(2-gt_sire)*(0.25*(2-gt_siredam)+0.5*(1-p_maf))
    p_Qq           <- 1-(p_QQ+p_qq)
    gt_matrix[i,3] <- sample(c(2,1,0),1,prob=c(p_QQ,p_Qq,p_qq))
    gt_matrix[i,4] <- as.numeric(p_QQ-p_qq)    # add = p(QQ)-p(qq)
    gt_matrix[i,5] <- as.numeric(p_Qq)         # dom = p(Qq)
   }
  # exporting from function
  gt_matrix<<-gt_matrix
}
```

```r
analyze <- function (h2,domratio,a)
{
  library(car)
  library(multcomp)

  s2P    <- 1000000       # Set is a phenotypic sd of 1000.
  s2A    <- h2*s2P        # genetic variance given h2
  sA     <- sqrt(s2A)     # additive genetic variance
  sE     <- sqrt(s2P-s2A) # residual variance
  a_QTL <- a*sA           # additive genetic effect given a.

  # QTL effects for the three states
  qtl_effect    <- array(0,dim=c(3))
  qtl_effect[1] <- 0
  qtl_effect[2] <- (1+domratio)*a_QTL
  qtl_effect[3] <- 2*a_QTL

  # Generating phenotypes (y) based on "true" genotypes (in gt_matrix[,3]).
  y <- 8000 + qtl_effect[gt_matrix[,3]+1] + rnorm(nrow(gt_matrix),0,sE)

  # ANOVA & contrast tests with 'true' genotypes
  test <- data.frame(gt=as.numeric(paste(gt_matrix[,3])),y=y)
  test[test["gt"]=="2","gt"] <- "AA"
  test[test["gt"]=="1","gt"] <- "AC"
  test[test["gt"]=="0","gt"] <- "CC"
  test[,"y"]  <- as.numeric(test[,"y"])
  test[,"gt"] <- as.factor(test[,"gt"])
  amod <- aov(y ~ gt, data = test)
  contrast_add <- summary(glht(amod, linfct = mcp(gt = c(1/2,0,-1/2))))
  contrast_dom <- summary(glht(amod, linfct = mcp(gt = c(-1/2, 1, -1/2))))

  # Regression with additive and dominance model (Haley & Knott F2)
  # with add_reg = p(QQ)-p(qq) and dom_reg = p(Qq).
  add_reg     <- gt_matrix[,4]
  dom_reg     <- gt_matrix[,5]
  fit_snp     <- lm(y~add_reg + dom_reg)

  # exporting the data
  summary_reg  <<- summary(fit_snp)
  contrast_add <<- contrast_add
  contrast_dom <<- contrast_dom
}

# ------------------------------------------------------------------
# |         2. Multimarker-scenario with related cows              |
# ------------------------------------------------------------------
# simulate(n_cow,n_snp,heri,ratio_qtl,add_ratio,daughters)
#   Simulates the multi-marker szenario with related cows.
#   Given are the number of cows, number of markers, heritability,
#   variance fraction explained by the markers, the ration of
#   additive variance and the number of decendents per sire/grandsire.
#
#   Exports and writes to disk:
#     sires & grandsires (ID, phenotype,genotype)
#     cows (ID, phenotype, sire, grandsire, n_marker*[genotype,add,dom])
#     marker_data (ID, explained variance, variance add, variance dom,
#     MAF, effect add, effect dom, is_QTL)
#
#   call: simulate(10000,100,0.3,0.5,0.5,50)



simulate <-function (n_cow,n_snp,heri,ratio_qtl,add_ratio,daughters)
{

  # define heriatbility and phenotypic variance
  Vp <- 1000000
  h2 <- heri

  # additive genetic variance
  Va <- Vp*h2

  # residual variance
  Ve <- Vp-Va
```

```
# Definition of QTL - variance as fractions of Va,
# for example here big effect 0.8*Va, small 0.2*Va,
# and 90% of the markers are small, 10% are strong.
Vqtl_all  <- Va*ratio_qtl      # variance explained by all markers
Vpoly     <- Va-Vqtl_all       # polygenic variance (the rest)
snp_big   <- rep(0,n_snp*0.1)
snp_big   <- snp_big+(0.8*Vqtl_all/length(snp_big))
snp_small <- rep(0,n_snp*0.9)
snp_small <- snp_small+(0.2*Vqtl_all/length(snp_small))

# Creation of an object for the markers, with columns for
# QTL-ID, QTL-variance, add & dom variances, MAF, add & dom effect,
# "QTL" as significance identifier (used later).
marker    <- array(0,dim=c(n_snp,8))
colnames(marker) <- c("ID","VaQTL","Vadd","Vdom","p","a","d","QTL")
marker[,1] <- seq(1,n_snp,by=1)
# sampling the markers (see above)
marker[,2] <- sample(c(snp_big,snp_small),n_snp,,replace=F) ## neu
# defining additive variance
marker[,3] <- add_ratio*marker[,2]
# defining dominance variance
marker[,4] <- marker[,2]-marker[,3]
# sampling MAFs for the markers
marker[,5] <- sample(seq(0.05,0.95,by=0.05),n_snp,prob=rep(0.1,19),replace=T)
# estimating dom effect
marker[,7] <- sqrt(marker[,4])/(2*marker[,5]*(1-marker[,5]))
# estimating additive effect
marker[,6] <- sqrt((marker[,3]/(marker[,5]*(1-marker[,5]))))-(marker[,7]*((1-marker[,5])-marker[,5]))
# boolean for 'is QTL'
marker[marker[,2] != snp_big,8]<-0
marker[marker[,2] == snp_big,8]<-1

# creation of objects for sires and grandsires
sires <- array(0,dim=c(n_cow/daughters,2+n_snp))
snips <- c()
for(i in 1:n_snp)
{
  snips <- c(snips,paste("GT",i,sep="_"))
}
colnames(sires)      <- c("ID","BV",snips)
grandsires           <- array(0,dim=c(n_cow/daughters,2+n_snp))
colnames(grandsires) <- c("ID","BV",snips)
sires[,1]            <- seq(1,nrow(sires),by=1)
grandsires[,1]       <- seq(1,nrow(grandsires),by=1)

# assigning breeding values
if (Vpoly>0)
{
  sires[,2]      <- rnorm(nrow(sires),0,sqrt(Vpoly))
  grandsires[,2] <- rnorm(nrow(grandsires),0,sqrt(Vpoly))
}else{
  sires[,2]      <- 0
  grandsires[,2] <- 0
}

# sampling of sires' and grandsires' genotypes, given HW-equilibrium
for (j in 1:nrow(sires))
{
  for(i in 1:n_snp)
  {
    freq_p            <- marker[marker[,"ID"]==i,"p"]
    freq_q            <- 1-freq_p
    sires[j,2+i]      <- sample(c(2,1,0),1,prob=c(freq_p**2,2*freq_p*freq_q,freq_q**2))
    grandsires[j,2+i] <- sample(c(2,1,0),1,prob=c(freq_p**2,2*freq_p*freq_q,freq_q**2))
  }
}

# creation of an object for the cows, colums for
# ID, yield, sire, grandsire, "true" genotype, deduced add & dom.
cows <- array(0,dim=c(n_cow,4+3*n_snp))
snps <- c()
for(i in 1:n_snp)
{
  snps <- c(snps,paste("GT",i,sep="_"),paste("add",i,sep="_"),paste("dom",i,sep="_"))
}
colnames(cows) <- c("ID","Yield","Vsire","GS",snps)
```

```
  cows[,1]       <- seq(1,n_cow,by=1)
  cows[,3]       <- sample(sires[,1],nrow(cows),replace=T)
  cows[,4]       <- sample(grandsires[,1],nrow(cows),replace=T)

  # assign genotypes and add & dom coefficients,
  # this step could easily be vectorized for n_markers
  #
  for (j in 1:n_cow)
  {
    cows[j,"Yield"] <- 0
    for (i in 1:n_snp)
    {
      p_maf <- marker[i,5]
      # calling the sire's and grandsires GT
      gt_sire    <- sires[sires[,"ID"]==cows[j,"Vsire"],paste("GT",i,sep="_")]
      gt_siredam <- grandsires[grandsires[,"ID"]==cows[j,"GS"],paste("GT",i,sep="_")]
      # calculating the genotype probabilities
      p_QQ <- 0.5*gt_sire*(0.25*gt_siredam+0.5*p_maf)
      p_qq <- 0.5*(2-gt_sire)*(0.25*(2-gt_siredam)+0.5*(1-p_maf))
      p_Qq <- 1-(p_QQ+p_qq)
      # sample 'true' genotype and assign the coefficients
      cows[j,paste("GT",i,sep="_")]  <- sample(c(2,1,0),1,prob=c(p_QQ,p_Qq,p_qq))
      cows[j,paste("add",i,sep="_")] <- as.numeric(p_QQ-p_qq)
      cows[j,paste("dom",i,sep="_")] <- as.numeric(p_Qq)
      # effects, calculated based on the 'true' genotypes
      qtl_effect      <- array(0,dim=c(3))
      qtl_effect[1]   <- 0
      qtl_effect[2]   <- marker[i,"a"]+marker[i,"d"]
      qtl_effect[3]   <- 2*marker[i,"a"]
      cows[j,"Yield"] <- cows[j,"Yield"]+qtl_effect[cows[j,paste("GT",i,sep="_")]+1]
    }
    # addition of a residual and a polygenic effect, based on the breeding values
    cows[j,"Yield"] <- cows[j,"Yield"]+0.5*sires[sires[,1]==cows[j,3],2]
    +0.25*grandsires[grandsires[,1]==cows[i,4],2]
  }

  # add residual polygenic effect
  cows[,2] <- cows[,2]+0.25*(rnorm(1,0,sqrt(Vpoly)))
  # add residual effect
  cows[,2] <- cows[,2]+rnorm(nrow(cows),0,sqrt(Ve))

  # convert to data frames and exporting
  sire        <- as.data.frame(sires)
  grandsire   <- as.data.frame(grandsires)
  cows        <- as.data.frame(cows)
  sires       <<- sires
  grandsires  <<- grandsires
  cows        <<- cows
  marker_data <<- as.data.frame(marker)

  # write on disk
  write.table(sire,file="sires.txt",sep="\t",row.names=F,quote=F)
  write.table(grandsire,file="grandsires.txt",sep="\t",row.names=F,quote=F)
  write.table(cows,file="cows.txt",sep="\t",row.names=F,quote=F)
  write.table(marker_data,file="marker_data.txt",sep="\t",row.names=F,quote=F)
}



#########################################################################
#       B.) Shell- and R-scripts used for real data analysis            #
#########################################################################

# ------------------------------------------------------------------------------------------
# |   1. make_regressors.R: R-script to deduce GT-probabilities and derive coefficients     |
# ------------------------------------------------------------------------------------------

#
# functions in R to deduce genotype probabilities and
# calculate additive and dominance regressors to detect
# dominance effects in paternally genotyped populations
# of dairy cows.
#
# Version 0.72 - june 05 2012, T.J.Boysen
# ___
#
```

```
#  This R-script calculates the coefficients (regressors) for additive and dominance.
#
#   This function needs:
#     - A list of bulls in form of a tfam file (bulls.tfam)
#
#     - The pedigree info of the cows in a file with the colums [ ID sire damsire ],
#       header and tab delimiter (pedinfo_sample.csv).
#
#     - A map of the markers, without header and tab delimiter, 4 colums:
#       Chr SNP cM MB  (the third is not used).
#
#     - The genotypes of the bulls als plink-Binary (bulls.bed, bulls.bim), from these
#       files the genotypes for one chromosome are extracted and processed by an external
#       shellscript. This is explained below.
#
#    To speed things up the preparation of genotype data and some conversions are done
#   externally. For this a shellscript (runplink.sh) is started by system calls.
#   The script calls the genome analysis toolset "plink" (PURCELL et al. 2007) to
#   extract genotypes for one chromosome at a time, recode to the [1,2] notation to
#   simplify further steps and calculate allele frequencies.
#    This process leaves the files ChrXX.tped, ChrXX.tfam, ChrXX.frq and ChrXX.map (with
#   XX: the number of the chromosome) in the working directory. The sequences of animals
#   and markers are defined by these files.
#    Further conversions are done in this script by the stream editor sed.
#   The shell script is included in this file below after the R part.
# ___
#
# Usage:
#   By running this script the needed functions and environment variables are implemented.
#   make_reg(14) calculates and saves the add and dom regressors for chr.14 to the disk.


### init ###
# Static Definitions
 rm(list=ls());gc()         # rm -rf *  <!>
 oncluster    <- FALSE      # If true the settings for a remote machine are used.
 OS           <- "linux"    # 'linux' oder 'win' - used to switch directories - but the needed
                            # (shell-)scripting is implemented for linux/unix only.
 RAM          <-  16        # Avaliable RAM [GB].
 Mode         <- "static"   # Static filenames, no select file dialogs during run.
 debug        <- FALSE      # Activates debug information --v

# Input files
 samplefile  <- "pedinfo.csv"      # Sample of cows
 mapfile     <- "Bovin50k.map"     # map (SNP-names and Position)
                                   # The genotype files are defined in the external shellscript

# Working directories, adapt this
 if (oncluster==FALSE)
  {
   if (OS=="win")   setwd("D:/user/")
   if (OS=="linux") setwd("/home/user/")
  }
 if (oncluster==TRUE)
  {
   setwd ("/server/user")  # home dir on remote machine
   RAM   <- 256
  }

# Flags - volatile, don't change here
 plinkcall    <- FALSE # Flag deactivates messages in import_chr()
 aktChr       <- 0     # Holds the current Chromosome.


### functions ###

make_reg <- function(CHR)
{
 print(paste("Regressors add & dom are calculated for Chr.",CHR,".",sep=""))
 print(paste("Used are the cows in the sample file '",samplefile,"'.",sep=""))
 if (oncluster == TRUE)
 {
  print("Results are saved in parts (oncluster=TRUE).")
 } else {
  print("Results are saved in one file.")
 }
```

```
print("")
print("Preparing genotypes (Plink binary file)...")
plink(CHR) # plink-routine is called here to extract/convert the genotypes.
print("");print("Import of genotypes and pedigree-info...")
tfamfile <- paste("Chr",as.character(CHR),".tfam",sep="")
freqfile <- paste("Chr",as.character(CHR),".frq",sep="")
tpedfile <- paste("Chr",as.character(CHR),".tped",sep="")
bullen   <- read.table(tfamfile, sep="", header=F, stringsAsFactors=F,
                       na.strings="NA", col.names=c("FAM", "ID", "SIRE", "DAM", "SEX", "PHENO"))
if ( Mode=="static")
{
 print(paste(" Reading Pedigree sample (",samplefile,")...",sep=""))
 kuehe   <- read.table(file=samplefile, sep="", header=T, stringsAsFactors=F, na.strings="NA")
} else {
 print("  Choose File with pedigree information [ID,sire,dam-sire] (pedinfo_sample.csv)...")
 kuehe   <- read.table(file.choose(), sep="", header=T, stringsAsFactors=F, na.strings="NA")
}
if ( length(unique(kuehe[,1])) != dim(kuehe)[1]) print("ERROR: duplicate IDs, has to be corrected.")
frequenz <- read.table(freqfile, sep="", header=T, stringsAsFactors=F)
tierzahl                    <- length(bullen[,2])              # header in freq:
recoded_A1Seq               <- seq(1,tierzahl*2,2)             # CHR, SNP, A1, A2, MAF, NCHROBS
recoded_A2Seq               <- seq(2,tierzahl*2,2)
recoded_cols                <- vector(mode="character", 2*tierzahl)
recoded_cols[recoded_A1Seq] <- bullen[,2]
recoded_cols[recoded_A2Seq] <- bullen[,2]                      # header for Allel 1 & 2.
recoded_cols[recoded_A1Seq] <- paste(recoded_cols[recoded_A1Seq], "_1", sep="")
recoded_cols[recoded_A2Seq] <- paste(recoded_cols[recoded_A2Seq], "_2", sep="")
recoded_colnames            <- append(c("CHR", "SNP", "NUL", "POS"),recoded_cols)
recoded  <- read.table(tpedfile, header=F, stringsAsFactors=F, na.strings="NA")
colnames(recoded) <- recoded_colnames
snpzahl <- length(frequenz[,3])
if ( ( (ncol(recoded)-4) /2 ) == tierzahl )
{
 if ( dim(recoded)[1] == snpzahl )
 {
  print(" The data was read sucessfully:")
  print(paste("   Bulls   in tfam: ",as.character(tierzahl)),sep="")
  print(paste("   Markers in freq: ",as.character(snpzahl)),sep="")
  print(paste("   Animals in tped-Datei: ",as.character((dim(recoded)[2]-4)/2),sep=""))
  print(paste("   Markers in tped-Datei: ",as.character(dim(recoded)[1])),sep="")
 }
} else
 {
  print("There's something wrong with the data:")
  print(paste("   Bulls   (tfam): ",as.character(tierzahl)),sep="")
  print(paste("   Markers (freq): ",as.character(snpzahl)),sep="")
  print(" aber:")
  print(paste("   Animals in tped-Datei: ",as.character((dim(recoded)[2]-4)/2),sep=""))
  print(paste("   Markers in tped-Datei: ",as.character(dim(recoded)[1])),sep="")
 }
rm(list=c("recoded_A1Seq", "recoded_A2Seq","recoded_cols", "recoded_colnames"))
#gc() # Garbage collection
print(" ")                              # Merging the relevant data in a structure to
print("Now converting the data...")     # allow a much more efficient coding than loops.
snpinfo <- frequenz[,c(2,3,5)]
colnames(snpinfo) <- c("SNP","MIN","MAF")
 #  The conversion of the allelea in p(A) is done externally with stream editor sed:
 #  sed -e's/1/0.5/g';'s/2/0/g'
 #   - Minor-Allel ist always the "1" im case of the [1,2]-coding used by plink.
 #   - If "1" then set 0.5, if "2" dann 0.
 #   - The sum of the two values is p(A).
rec_sed  <- recoded[,5:ncol(recoded)]   # getting the columns with the alleles.
suffix <- round(runif(1,0,1000000))     # To prevent collisions: random file name!
tempname <- paste("rec_sed_",as.character(suffix),".tmp",sep="")    # save data for conversion
write.table(rec_sed, file=tempname, quote=F, sep="\t", row.names=F, col.names=F)
call <-paste("./runplink.sh ",tempname," probs",sep="")
null <- system(call,intern=T)           # call of runplink to do the conversion.
rec_sed  <- read.table(tempname,sep="\t", header=F, stringsAsFactors=F)  # reread converted data
call <-paste("./runplink.sh ",tempname," gc",sep="")                     # dispose of tempdata.
null     <- system(call,intern=T)
geno     <- cbind(recoded[,1:4],rec_sed, stringsAsFactors=F)
colnames(geno) <- colnames(recoded)
rm(rec_sed);rm(recoded)
geno     <-cbind(geno[,c(2,1)],geno[,3:ncol(geno)],snpinfo[,2:3])
print("")
print("Clean up genotypes...")
```

```
totalsnp <- nrow(geno)
print(paste("    Number of Markers:  ",as.character(totalsnp)))
geno     <- geno[geno[,6+2*tierzahl]!=0,]                          # SNPs mit MAF=0 are removed.
print(paste("    Markers with MAF=0: ",as.character(totalsnp-nrow(geno))))
print(paste("    Remaining Markers:  ",as.character(nrow(geno))))
header   <- colnames(geno)
write.table(geno[,5:ncol(geno)], file=tempname, quote=F, sep="\t", row.names=F, col.names=F)
call <- paste("./runplink.sh ",tempname," NAs",sep="")
null     <- system(call, intern=T)                                 # -9 to NA (s/-9/NA/).
geno[,5:ncol(geno)]<- read.table(file=tempname,sep="\t", header=F, stringsAsFactors=F)
call <-paste("./runplink.sh ",tempname," gc",sep="")               # dispose of tempdata.
null     <- system(call,intern=T)
snpfile  <- paste("Chr",as.character(CHR),"_snp.csv",sep="")                  # The final
write.table(geno[,c(1,4)],snpfile, quote=F, sep="\t", row.names=F, col.names=T) # marker sequence.
print(paste("The new marker list was saved in ",snpfile,".", sep=""))
geno     <- geno[-3]  # removing unused column (cM).
    #  Object 'geno' is now complete. It has the following columns:
    #  [SNP CHR POS ID1_1 ID1_2 ID2_1 ID2_2 ... MIN MAF]
    #  with allele values of 0 (major allele) and 0.5 (minor allele).
print("")
print("Now recoding to p(A), this takes some time...")  # Now the object 'prob' is created.
snpzahl  <- nrow(geno)                                  # Number of SNP after clean-up.
tierzahl <- (ncol(geno)-5)/2                            # Number of Animals (if there is filtering)
prob     <- as.data.frame(matrix(0,nrow = snpzahl, ncol = tierzahl+3))
colnames(prob)[1:3] <- c("SNP","CHR","MAF")
colnames(prob)[4:ncol(prob)] <- bullen[,2]
prob[,1:3] <- geno[,c(1,2,ncol(geno))]                  # Registering SNPs with CHR und MAF.
for (tier in 1:tierzahl)
{
 prob[,(3+tier)] <- geno[,(2+tier*2)]+geno[,(3+tier*2)] # Registering Sums of allele values.
}
rm(geno);#gc()                                          # Garbage collection, geno is redundant.
print("done.")
print("Concluding tests before starting the calculations...") # Let's double-check...
F2ped <- kuehe[!is.na(kuehe[,3]),]                      # Only cows, where sire && dsire known.
F2ped <- F2ped[!is.na(F2ped[,2]),]
if (nrow(kuehe) == nrow(F2ped))
 {
 print("  Pedigree is OK: For each ID sire & damsires are known.")
 } else print(paste("  ",as.character(dim(kuehe)[1]-dim(F2ped)[1]),
              " cows with missing sire/damsire were removed.",sep=""))
colnames(F2ped) <- c("ID","SIRE","DSIRE")
F2ped$SIRE<-as.character(F2ped$SIRE)                    # cast LOMs to Characters
F2ped$DSIRE<-as.character(F2ped$DSIRE)
rownames(F2ped) <- F2ped[,1]                            # copy IDs in rownames
print("  Allel-probabilities:")
tiereinprob <-colnames(prob)[-c(1,2,3)]
tiereausped<-unique(c(F2ped$SIRE,F2ped$DSIRE))
if (length(setdiff(F2ped$SIRE,tiereinprob))==0 && length(setdiff(F2ped$DSIRE,tiereinprob))==0)
 {
  print("  Information for all sires und damsires are avaliable,")
  print(paste("  additional for ",as.character(length(setdiff(tiereinprob,tiereausped))),
              " Bulls.",sep=""))
  print("Everything seems OK. Now starting the calculations")
 } else
 {
  print("ERROR! There are missing informations in 'prob':")
  print(paste("  ",as.character(length(setdiff(F2ped$SIRE,tiereinprob))),
        " sires & ",as.character(length(setdiff(F2ped$DSIRE,tiereinprob))),
        " damsires without data.",sep=""))
  if (length(setdiff(F2ped$SIRE,tiereinprob))!=0)
  {
   sires.def<<-setdiff(F2ped$SIRE,tiereinprob)
   print("Affected sires are listed in 'sires.def'.")
  }
  if (length(setdiff(F2ped$DSIRE,tiereinprob))!=0)
  {
   dsires.def<<-setdiff(F2ped$DSIRE,tiereinprob)
   print("Affected sires are listed in 'sires.def'.")
  }
  print("Stopping here.")
  BREAK  # forced break
 }
rm(kuehe); #gc()
```

```
 # Now we have:
 #
 #   prob   with the LINES  SNP, CHR, MAF
 #          and therein Bull_1...Bulle_n       - This is the source for p-Values.
 #
 #   F2ped  with COLUMNS  ID SIRE DSIRE,
 #          rowname holds the cow's IDs.       - This is the source for cows and is template
 #                                               for the sequence of the data in the now to
 #                                               build 3D-object 'probarray'.
 #
 # Building the object 'probarray'
 #   The base is F2ped, next dimension is Number of markers in "prob":
 #   probarray[F2-Animal, Level, Marker],
 # [F2-Animal, , ] : Animals in the same sequence as F2ped.
 # [   ,Level,   ] : Level 1:p(sire), 2:p(dsire), level 3 holds intermediate value f(A)+p(A)dsire,
 #                   levels 4-6 are p(AA), P(AB) and P(BB), regressor for additive is in level 7,
 #                   dominance counterpart is P(AB) in level 5. Without contrast here,
 #                   with contrast it would be dom = p(AB) - 0.5(p(AA)+p(BB)).
 # [  ,   ,Marker] : The sequence is coming from column SNP in 'prob'.
 #
 # The calculation of the coefficients is done by "salami tactics". If there is insufficient RAM
 # temporary objects with parts of the cows are processed. Here's the point to parallelizing the
 # whole thing in C.
 # As treshold RAM/4 is set here, because also the results are using more and more RAM.
 # In case of bigger sample sizes it's better so save these parts and free the RAM and
 # to assemble them later on the shell. This is done if flag "oncluster" is set.
 #
if ( debug == TRUE ) # just some debug infos are written to disk.
{
 probfile<-paste("Chr",as.character(CHR),"_prob.csv",sep="")
 print("### debug-Informationen ###")
 print(paste("Allel-Herkunftswahrscheinlichkeiten ('prob') werden in ",
       probfile," gesichert.",sep=""))
 write.table(prob, file=probfile, quote=F, sep="\t", row.names=F, col.names=T)
}
print(paste("  To be processed are ",as.character(nrow(F2ped)),
      "animals with ",as.character(nrow(prob)),"markers each."))
print("Testing RAM...")
probsize   <- round((nrow(F2ped)*nrow(prob)*64)/1073741824,0)
if ( probsize > RAM/4 ) # Given more than 200,000 Animals there could be a problem with 16GB RAM
{                       # because keeping the complete results costs also memory.
 ### Begin interval processing ###     (the salami slices)
 print("  RAM shortage is possible, teste interval processing ...")
 Teile     <- round(probsize/(RAM/8),0)
 probarray <- array(NA, dim=c(nrow(F2ped)/Teile,7,nrow(prob))) # teste mal...
 print(paste("  OK. Calculating in ",as.character(Teile)," Parts of ",
       as.character(round(as.numeric(object.size(probarray)/1073741824),1))," GB",sep=""))
 nF2       <- dim(F2ped)[1]
 Int       <- round((nF2/Teile),0)
 IntList   <- array(0, dim=c(Teile,2))
 colnames(IntList)=c("start","stop")  # The list of start and stop positions are defined here.
 for ( i in 0:(Teile-2))
 {
  IntList[i+1,] <- c((i*Int)+1,(i+1)*Int)
 }
 IntList[i+2,]  <- c(((i+1)*Int)+1,nF2)
 print(" ")
 if ( debug == TRUE )                 # For debugging purposes these positions can be saved here,
 {                                    # it's possible to continue after a mishap.
  print("  The Positions in 'F2ped' for the intervall processing")
  print(paste("  are saved in the file 'koeff_part_ChrXX.log'.",sep=""))
  print(" ")
  koeexp <- paste("koeff_part_Chr",as.character(CHR),".log",sep="")
  write.table(IntList, file=koeexp, quote=F, sep="\t", row.names=F, col.names=T)
 }
 print("- Start of the calculations -")
 rm(list=c("probarray","i","nF2","Int"));gc()
 addexp <- paste("Chr",as.character(CHR),"_add.csv",sep="") # File with results for additive
 domexp <- paste("Chr",as.character(CHR),"_dom.csv",sep="") # File with results for dominance
 for ( Teil in 1:Teile )
  {                                   # Now the parts are processed consecutively.
  print(" ")
  print(paste("Processing part ",as.character(Teil),"/",as.character(Teile),sep=""))
  probarray <- array(NA,dim=c(as.numeric(IntList[Teil,2])
                              -as.numeric(IntList[Teil,1])+1,8,nrow(prob)))
  F2part    <- F2ped[as.numeric(IntList[Teil,1]):as.numeric(IntList[Teil,2]),]
```

```
 print("  Preparing data...")
 # List of frequencies p(A) is prob$MAF
 # List of prob. is prob[[tier]]
 # sire/dsire of cow is F2ped[F2ped[,1]==cow,2/3]
 for ( i in 1:nrow(F2part))                # For all cows in this part:
  {
   probarray[i,1,] <- prob[[F2part[i,2]]]  # Vectors with p(A) for  sire in level 1
   probarray[i,2,] <- prob[[F2part[i,3]]]  # Vectors with p(A) for dsire in level 2
  }
if ( debug == TRUE )
 {
  testcow <- 1   # If there are problems with all these assignments: Lets follow a
  print("")      # single cow and dump the information here.
  print("### debug-informationen ###")
  print("## F2part: cow, sire ,dsire ##");print(as.character(F2part[testcow,]))
  print("## prob from sire ##");print(as.character(prob[[as.character(F2part[testcow,])[2]]]))
  print("## prob from dsire ##");print(as.character(prob[[as.character(F2part[testcow,])[3]]]))
  print("## probarray cow: Level 1 (sire) ##");print(as.character(probarray[testcow,1,]))
  print("## probarray cow: Level 2 (dsire) ##");print(as.character(probarray[testcow,2,]))
  print("")
 }
 print("  Calculating the coefficients...")       # Operation by-level in the 3D-object
 probarray[,3,] <- matrix(mapply("+",t(probarray[,2,]),prob$MAF),
                   ncol=nrow(prob), nrow=nrow(F2part), byrow=T)
 probarray[,4,] <- (0.5*probarray[,3,])*probarray[,1,]                        # p(AA)
 probarray[,5,] <- (probarray[,1,]*(1-probarray[,3,]))+0.5*probarray[,3,]     # p(AB)       = dom
 probarray[,6,] <- (1-probarray[,1,])*( 1-(0.5*(probarray[,3,])))             # p(BB)
 probarray[,7,] <- probarray[,4,]-probarray[,6,]                              # p(AA)-p(BB) = add

 if (oncluster==TRUE)                              # if(oncluster) assemble just the current part
 {
   add <- t(matrix(data=probarray[,7,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
            dimnames=list(rownames(F2part),prob$SNP)))
   dom <- t(matrix(data=probarray[,5,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
            dimnames=list(rownames(F2part),prob$SNP)))
 }
 else                               # if(!oncluster) assemble the calculated part with prior ones.
 {                                  # But: With more than a few hundred thousand cows this consumes
   print("  Data assembly...")      # a lot of RAM. With more than 200,000 cows and 16GB this is
   if ( Teil == 1)                  # nonsence to do in R.
    {
     add <- t(matrix(data=probarray[,7,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
              dimnames=list(rownames(F2part),prob$SNP)))
     dom <- t(matrix(data=probarray[,5,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
              dimnames=list(rownames(F2part),prob$SNP)))
     addfull <- add
     domfull <- dom
    }
   else
    {
     add <- t(matrix(data=probarray[,7,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
              dimnames=list(rownames(F2part),prob$SNP)))
     dom <- t(matrix(data=probarray[,5,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
              dimnames=list(rownames(F2part),prob$SNP)))
     addfull <- cbind(addfull,add)
     domfull <- cbind(domfull,dom)
    }
 }
 rm(probarray);gc()              # Saving the part or the whole data, free the memory.
 if (oncluster == TRUE)
  {
   print("  Save current part...")
   addprt<-paste("Chr",as.character(CHR),"_add_",as.character(Teil),".csv",sep="")
   domprt<-paste("Chr",as.character(CHR),"_dom_",as.character(Teil),".csv",sep="")
   write.table(add, file=addprt, quote=F, sep="\t", row.names=T, col.names=T)
   write.table(dom, file=domprt, quote=F, sep="\t", row.names=T, col.names=T)
   rm(list=c("add","dom"));gc()     # free memory.
  }
  else
  {
   rm(list=c("add","dom"));gc()     # free memory.
   print("  Exportiere bisherige Gesamtdaten...")
   write.table(addfull, file=addexp, quote=F, sep="\t", row.names=T, col.names=T)
   write.table(domfull, file=domexp, quote=F, sep="\t", row.names=T, col.names=T)
   print("Status:")
   print(paste(as.character(dim(addfull)[2])," Tiere",sep=""))
```

```
    }
  } # endfor Teil - End of the loop parts
  rm(Teil);rm(F2part);
  ### End of intervall processing ###
  } else {
  ### Begin complete processing ###
  print(" ")    # If there is sufficient RAM the same as above is done. But without
                # all that stuff around the parts.
  probarray <- array(NA, dim=c(nrow(F2ped),7,nrow(prob)))
  print("There seems to be sufficient RAM, the object 'probarray'")
  print(paste("needs ",as.character(round(as.numeric(object.size(probarray)/1048576),1)),
              " MB RAM.",sep=""))
  for ( i in 1:nrow(F2ped))
   {
    probarray[i,1,] <- prob[[F2ped[i,2]]]
    probarray[i,2,] <- prob[[F2ped[i,3]]]
   }
  if ( debug == TRUE )
   {
    testcow <- 1
    print("")
    print("### debug-informationen ###")
    print("## F2part: cow, sire ,dsire ##");print(as.character(F2part[testcow,]))
    print("## prob from sire ##");print(as.character(prob[[as.character(F2part[testcow,])[2]]]))
    print("## prob from dsire ##");print(as.character(prob[[as.character(F2part[testcow,])[3]]]))
    print("## probarray cow: Level 1 (sire) ##");print(as.character(probarray[testcow,1,]))
    print("## probarray cow: Level 2 (dsire) ##");print(as.character(probarray[testcow,2,]))
    print("")
   }
  print("- Start of the calculations -")
  probarray[,3,] <- matrix(mapply("+",t(probarray[,2,]),prob$MAF),
                    ncol=nrow(prob), nrow=nrow(F2ped), byrow=T)
  probarray[,4,] <- (0.5*probarray[,3,])*probarray[,1,]                    # p(AA)
  probarray[,5,] <- (probarray[,1,]*(1-probarray[,3,]))+0.5*probarray[,3,]    # p(AB)        = dom
  probarray[,6,] <- (1-probarray[,1,])*( 1-(0.5*(probarray[,3,])))          # p(BB)
  probarray[,7,] <- probarray[,4,]-probarray[,6,]                          # p(AA)-p(BB)  = add
  add <- t(matrix(data=probarray[,7,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
           dimnames=list(rownames(F2ped),prob$SNP)))
  dom <- t(matrix(data=probarray[,5,], ncol=dim(probarray)[3], nrow=dim(probarray)[1],
           dimnames=list(rownames(F2ped),prob$SNP)))
  print("Exporting the data...")
  addfile<-paste("Chr",as.character(CHR),"_add.csv",sep="")
  domfile<-paste("Chr",as.character(CHR),"_dom.csv",sep="")
  write.table(add,addfile,quote=F,sep="\t",row.names=T,col.names=T)
  write.table(dom,domfile,quote=F,sep="\t",row.names=T,col.names=T)
  }
  ### End of complete processing ###
 if ( debug == TRUE ){add<<-add;dom<<-dom}  # If debugging leave everything behind.
 print(paste("additive and dominance regressors (Chr.",as.character(CHR),") are ready..",sep=""))
 print("__done__")
}


# This function is called from make_coeff().
plink <- function(CHROM)
{
 if (OS=="linux")
  {
    plinkcall <<- TRUE
    import_chr(CHROM)  # Extraction to tfam/tped in [1,2] notation for one chromosome
    print(paste(" Allele frequencies are calculated... (Results in  Chr",
                as.character(CHROM),".frq)",sep=""))
    call<-paste("./runplink.sh ",as.character(CHROM)," freq",sep="")
    null<-system(call,intern=T)
    print("Done.")
  } else print(" Sorry - not implemented in Windows so far.")
}

import_chr <- function(CHROM) # This function is called by plink(). For the given chromosome
{                             # the map and the genotypes are imported and some tests are done.
 print(paste(" Importing map and genotypes for Chr.",as.numeric(CHROM),sep=""))
 if (OS=="linux")
 {
  alldata     <<- TRUE   # We are using the binary genotype file, so Flag up.
  aktChr      <<- CHROM  # Setting the active Chromosome
  error_flag  <- FALSE
```

```
  print(" Reading map...")
  map          <- read.table(file=mapfile, sep="", header=F, stringsAsFactors=F,
                  col.names=c("CHR", "SNP", "CM", "POS"), colClasses = "character")
  mapChr       <- map[map[,1]==CHROM,]
  mapChr       <- mapChr[order(as.numeric(mapChr[,4])),]
  markerliste <- mapChr[,2]
  nMap         <- length(markerliste)
  if (length(markerliste)!=length(unique(markerliste))) print("ERROR! Duplicates in the map.")
  print(paste(" According to the map ",as.character(nMap)," markers are on Chr.",
              as.character(CHROM),sep=""))
  print(paste(" The genotypes are extracted to 'Chr",as.character(CHROM),".tped' and 'Chr",
              as.character(CHROM),".tfam'...", sep=""))
  call         <- paste("./runplink.sh ",as.character(CHROM)," import",sep="")
  null         <- system(call,intern=T)    # plink reads from .bin & .bed and creates tfam tped.
  tfamfile     <- paste("Chr",as.character(CHROM),".tfam",sep="") # The created files are now read.
  tpedfile     <- paste("Chr",as.character(CHROM),".tped",sep="")
  bulls        <- read.table(tfamfile, sep="", header=F, stringsAsFactors=F,
                  col.names=c("FAM","ID","sire","dam","sex","pheno"), colClasses = "character")
  tped         <- read.table(tpedfile, sep="",header=F,stringsAsFactors=F,colClasses = "character")
  nSNP         <- nrow(tped)
  nVieh        <- nrow(bulls)
  ntped        <- (ncol(tped)-4)/2
  SNPliste     <- tped[,2]
  if (nVieh==ntped && nMap==nSNP)
  {
   print(paste(" It was read: tfam with ",as.character(nVieh)," bulls tped for ",
               as.character(nSNP)," Markers for ",as.character(ntped)," Animals.",sep=""))
  } else {
   if (nVieh!=ntped)
   {
    print(" ERROR! Numbers of animals not correct:"); error_flag <-TRUE
    print(paste("  tfam:  ",as.character(nVieh)," Tiere",sep=""))
    print(paste("  tped:  ",as.character(ntped)," Tiere,",sep=""))
   }
   if (nMap!=nSNP)
   {
    print(" WARNING: Number of SNP is not in accordance to the map.")
    print(paste("   map:  ",as.character(nMap)," Marker",sep=""))
    print(paste("  tped:  ",as.character(nSNP)," Marker.",sep=""))
    print(" Maybe the missing ones are filtered by plink (MAF<0.05)?")
   }
  }
  if (length(SNPliste)!=length(unique(SNPliste))){
      print("(import_chr): ERROR! Duplicate markers in tped-file.")
      error_flag <- TRUE
      }
  if (debug==TRUE) filteredSNP <<- setdiff(markerliste,SNPliste)
  if (length(setdiff(SNPliste,markerliste))!=0){
      print("(import_chr): ERROR! Markers in the tped-file are not defined in the map.")
      error_flag <- TRUE
     }
  if (nrow(bulls)!=length(unique(bulls$ID))){
      print("(import_chr): ERROR! Duplicate IDs in the tfam-file.")
      error_flag <- TRUE
      }
  print("")
  print(" Creating and saving current map...")
  mapChr  <- mapChr[mapChr$SNP %in% SNPliste,]
  bulls  <<- bulls
  mapChr <<- mapChr
  mapfile <- paste("Chr",as.character(CHROM),".map",sep="")
  print(paste(" Map saved as '",mapfile,"'.",sep=""))
  write.table(mapChr,file=mapfile, row.names=F, col.names=F, quote=F, sep="\t", dec=".")
  if ( plinkcall==TRUE)
  {
   print(" Import is completed."); print(""); plinkcall <<- FALSE
  } else {
   print(paste(" Data is avaliable as '",tfamfile,"', '",tpedfile,"' und ",mapfile,".",sep=""))
   print(" Object 'bulls' containes the IDs (tfam-format), 'mapChr' holds the map.")
   print(" _done_")
  }
 if( error_flag  == TRUE ){ print(" <<Correct errors first!>> ") }
 }
 else print(" Sorry - not implemented in Windows so far.")
}
```

```
# ------------------------------------------------------------------------
# |   2. runplink.sh: Shell-script called by make_regressors.R           |
# ------------------------------------------------------------------------

#!/bin/bash
#
# project dominance
# T.J.Boysen, V0.2  May 09 2012
# Routines called by make_regressors.R
# Calls plink and sed
# Arguments in UPN (arguments, operator)
# Argument1: Chromosome number or tempname if command is "probs"
# Argument2: command

# z.B. runplink 14 import
CHR=$1 ORDER=$2

# debug-Info
echo ' ------------------------------------ '
echo '  This is runplink, called with:'
echo '  argument '$CHR' & operator '$ORDER
echo ' ------------------------------------ '

# Import of genotypes from plink-binary,
# directory is here static: "./genotypes_bulls"
# and files are here static: bulls.bed, bulls.bim.
# See plink documentation for details.
if [ $ORDER == 'import' ]; then
 echo '<import> Importing and filtering genotypes f. BTA'$CHR' from binary, returning tfam/tped'
 cd genotypes_bulls
 ../plink --noweb --cow --nonfounders --allow-no-sex --bfile bulls --recode12 --chr $CHR --maf 0.05
          --transpose --missing-genotype NA --out Chr$CHR
 mv Chr$CHR.tped ../
 mv Chr$CHR.tfam ../
 rm Chr$CHR.*
 cd ..
fi

# calculating frequencies using plink and
# conversion of 'NA' to '-9'
if [ $ORDER == 'freq' ]; then
 echo ' <freq> Ableiten der Allelfrequenzen...'
 ./plink --noweb --cow --freq --nonfounders --allow-no-sex --missing-genotype NA --tped Chr$CHR.tped
         --tfam Chr$CHR.tfam --map Chr$CHRmap.map
 mv plink.frq Chr$CHR.frq
 if [ -f plink.hh ];then
  mv plink.hh Chr$CHR.hh
 fi
 if [ -f plink.log ];then
  rm plink.log
 fi
 # rm Chr$CHR.log
 echo 'Konvertiere NAs zu -9...'
 sed -e 's/NA/-9/g' Chr$CHR.tped > Chr$CHR.tped.temp
 mv Chr$CHR.tped.temp Chr$CHR.tped
 # rm Chr$CHR.tfam
 echo '__done__'
fi

# Routines called by make_reg():
# Replace [1,2] alleles to probabilities [0.5,0].
# In this case $1 is the temp file name.
#
if [ $ORDER == 'probs' ]; then
 echo ' <probs> Konvertiere zu Allelfrequenzen'
 sed -e 's/1/0.5/g' $1 > $1.temp
 sed -e 's/2/0/g' $1.temp > $1
 rm $1.temp
fi

if [ $ORDER == 'NAs' ]; then
 echo ' <NAs> Konvertiere -9 zu NA'
 sed -e 's/-9/NA/g' $1 > $1.temp
 mv $1.temp $1
```

```
fi

if [ $ORDER == 'gc' ]; then
 echo ' <gc> Entferne Temp-Datei'
 rm $1
 if [ -f plink.nosex ];then
  rm plink.nosex
 fi
fi

# ------------------------------------------------------------------------------
# |   3. Full_Model.R: R-script used for real data analysis applying the full model   |
# ------------------------------------------------------------------------------
#
# There are two files per chromosome, produced by make_regressors.R,
# with the additive and dominance dominance regressors for every cow.
# (ChrXX_add.csv, ChrXX_dom.csv). Those files include missing obser-
# vations due to missing genotypes in sires.
#
# Pedigree for animals with genotyped sire and grandsire
# is stored in the samplefile used in make_regressors.R and phenotypes
# are stored in phenofile wit columns ID, Trait1, Trait2 ..., both with
# tab delimiter and with header.
#

  library(asreml)
  setwd("/home/user")

  # Input files
  samplefile  <- "pedinfo.csv"       # Sample of cows
  phenofile   <- "F2-Cow-Data.csv"

  # Define trait (colname in phenofile) and chromosome,
  # number of bulls & cows
  trait  <- "YD_Fat.kg_L1"
  chr    <- 14
  nBulls <- 2081
  nCows  <- 542266

  # Read data and extract phenotypes for cows
  # with known sire and maternal grandsire
  ped    <- read.table(samplefile,header=T,sep="\t")
  f2     <- read.table(phenofile,header=T,sep="\t")
  f2_sub <- f2[is.element(f2[,1],ped[,1]),]

  # Get vector of phenotypes for the specified trait
  y <- f2_sub[,trait]

  # Create inverse of additve genetic relationship matrix.
  # Note the option 'mgs=T' indicating that the third column
  # represents the maternal grandsire rather than the dam.
  A     <- asreml.Ainverse(ped,mgs=T)
  A_inv <- A$ginv

  # Extract vector of all animals in pedigree
  animals <- A$pedigree$Kuh

  # Create data-frame for asreml-call. It includes two columns
  # for 'id' and 'y'. The first $nBulls rows represent sires/grand-
  # sires without phenotypes
  data     <- as.data.frame(cbind(animals,numeric(length(animals))))
  data[,2] <- as.numeric(data[,2])
  data[(nBulls+1):nrow(data),2]<-f2_sub[,trait]
  data[,1] <- as.factor(data[,1])
  data[,3] <- rep(0,nrow(data))
  data[,4] <- rep(0,nrow(data))

  # Sires/grandsires don't provide phenotypes or coefficients
  data[1:nBulls,2] <- NA
  data[1:nBulls,3] <- NA
  data[1:nBulls,4] <- NA
  colnames(data)   <- c("id","y","add","dom")

  # Get number of markers (first row is a header)
  call     <- paste("wc -l < Chr",as.character(CHR),"_add.csv",sep="")
  null     <- pipe(call)
```

```
  n_marker <- scan(null,what=numeric(0),quiet=T)-1
  close(null)
  rm(null)

  # Create array for results
  result <- array(0,dim=c(1,5))
  result <- as.data.frame(result)
  colnames(result) <- c("marker","p_add","p_dom","eff_add","eff_dom")

  # Exclude first column (rownames) of marker files
  call <- paste("cut -f 2- Chr",as.character(CHR),"_add.csv",sep="")
  null_add <- pipe(call)
  call <- paste("cut -f 2- Chr",as.character(CHR),"_dom.csv",sep="")
  null_dom <- pipe(call)

  # Start loop over all markers of chromosome
  for (i in 1:n_marker)
  {
   print(i)
   # Get vectors 'add' and 'dom' for all cows
   data[(nBulls+1):nrow(data),3] <- scan(file=null_add,skip=i,nlines=1,
                                         what=numeric(0),sep="\t",quiet=T)
   data[(nBulls+1):nrow(data),4] <- scan(file=null_dom,skip=i,nlines=1,
                                         what=numeric(0),sep="\t",quiet=T)
   # Run mixed model with fixed effects: intercept, add, dom ; random effect: animal
   mod <- asreml(y~add+dom,random=~ped(id,var=T),ginverse=list(id=A_inv),na.method.X="omit",
                 data=data,control=asreml.control(workspace=6e7))
   # Wald Test for fixed effects
   amod  <- wald.asreml(mod, ssType="conditional", denDF="numeric")
   coeff <- amod$Wald$Pr
   fix   <- mod$coefficients$fixed
   # Save results
   result[1,1] <- i                    # marker
   result[1,2] <- coeff[2]             # p-value add
   result[1,3] <- coeff[3]             # p-value dom
   result[1,4] <- as.numeric(fix[2])   # additive effect
   result[1,5] <- as.numeric(fix[1])   # dominance effect
   # Write and append results to the file
   write.table(result,file=paste(chr,trait,"full",sep="_"),col.names=F,row.names=F,
               quote=F,sep="\t",append=T)
  }
 print("__done__")

# ------------------------------------------------------------------
# |   4. Grammar.R: R-script used to do the 2-step analysis     |
# ------------------------------------------------------------------

# Input files as  mentioned above in Full_Model.R

  library(asreml)
  library(car)
  setwd("/home/user")

  # Input files
  samplefile  <- "pedinfo.csv"       # Sample of cows
  phenofile   <- "F2-Cow-Data.csv"

  # Define trait and chromosome, number of bulls & cows
  trait  <- "YD_Fat.kg_L1"
  chr    <- 14
  nBulls <- 2081
  nCows  <- 542266

  ped <- read.table(samplefile,header=T,sep="\t")
  f2  <- read.table(phenofile,header=T,sep="\t")

  # Extract phenotypes for cows with known sire and maternal grandsire
  f2_sub <- f2[is.element(f2[,1],ped[,1]),]

  # Get vector of phenotypes for the specified trait
  y <- f2_sub[,trait]

  # Create inverse of additve genetic relationship matrix.
  # Note the option 'mgs=T' indicating that the third column
  # represents the maternal grandsire rather than the dam.
  A     <- asreml.Ainverse(ped,mgs=T)
```

```
  A_inv <- A$ginv

  # Extract vector of all animals in pedigree
  animals <- A$pedigree$Kuh

  # Create data-frame for asreml-call. It includes two columns for 'id' and 'y'.
  # The first $nBulls rows represent sires/grandsires without phenotypes
  data     <- as.data.frame(cbind(animals,numeric(length(animals))))
  data[,2] <- as.numeric(data[,2])

  # Sires/Grandsires don't provide phenotypes
  data[1:nBulls,2] <- NA

  data[(nBulls+1):nrow(data),2] <- f2_sub[,trait]
  data <- cbind(data,data[,2],data[,2])
  colnames(data) <- c("id","y")
  data[,1] <- as.factor(data[,1])

  # Adjust phenotypes using animal model (random term). One fixed effect (intercept).
  mod<-asreml(y~1,random=~ped(id,var=T),ginverse=list(id=A_inv),data=data,
              control=asreml.control(workspace=6e7))

  # New phenotypes are the residuals of the mixed model
  y <- resid(mod)

  # Extract cows
  y <- y[(nBulls+1):length(y)]

# First step of Grammar-approach completed. Phenotypes are
adjusted and # residuals will serve as depending variable of an
Ordinary Least Squares approach.

# Step two...

  # Get number of markers (first row is a header)
  call     <- paste("wc -l < Chr",as.character(CHR),"_add.csv",sep="")
  null     <- pipe(call)
  n_marker <- scan(null,what=numeric(0),quiet=T)-1
  close(null)
  rm(null)

  # Create array for results
  result <- array(0,dim=c(1,5))
  result <- as.data.frame(result)
  colnames(result) <- c("marker","p_add","p_dom","eff_add","eff_dom")

  # Exclude first column (rownames) of marker files
  call <- paste("cut -f 2- Chr",as.character(CHR),"_add.csv",sep="")
  null_add <- pipe(call)
  call <- paste("cut -f 2- Chr",as.character(CHR),"_dom.csv",sep="")
  null_dom <- pipe(call)

  # Start loop over all markers of chromosome
  for (i in 1:n_marker)
  {
   print(i)
   # Get vectors 'add' and 'dom' for all cows
  add <- scan(file=null_add,skip=i,nlines=1,what=numeric(0),sep="\t",quiet=T)
  dom <- scan(file=null_dom,skip=i,nlines=1,what=numeric(0),sep="\t",quiet=T)
  # Ordinary Least Squares with subsequent Ancova
  mod  <- lm(y~add+dom)
  amod <- Anova(mod,type='III',singular.ok=T)
  # Save results
  result[1,1] <- i            # marker
  result[1,2] <- amod[2,4]    # p-value add
  result[1,3] <- amod[3,4]    # p-value dom
  result[1,4] <- as.numeric(mod$coefficients[2])    # additive effect
  result[1,5] <- as.numeric(mod$coefficients[3])    # dominance effect
  # Write results into file
  write.table(result,file=paste(chr,trait,"grammar",sep="_"),
              col.names=F,row.names=F,quote=F,sep="\t",append=T)
  }
 print("__done__")
```