# Heritability estimation using relationship matrices

Jing Hua Zhao
University of Cambridge

June 3, 2015

## 1 Introduction

Twin and family studies have been the standard approach for heritability estimation, where differences between monozygotic and dizygotic twin pairs are attributed to genetics and familial relationships are linked with a polygenic effect. Usually the estimate from twin studies is higher than that from family studies. It is difficult to tease out influence of the common environment for both types of data.

There has been a lot of interest recently in use of genomic relationship matrices (GRMs) regardless their famiilial background so unrelated individuals can also be used (Yang et al. (2010)). The GRM associated with a polygenic component in a random effects or mixed model mirrors the role of a relationship matrix based on family structures. A dedicated computer program called GCTA (genome-wide complex trait analysis) is available (Yang et al. (2011)). Work has been done to show the utility of GRM in linkage studies (Day-Williams et al. (2011)) and heritability estimation (Klimentidis et al. (2013)).

Here we use a very simple family to illustrate heritability estimation. As GRMs typically involve large quantity of genomic data, we will use the relationship matrix derived from the family structure as if it was a GRM. We then provide examples to read/write GRMs either in text or binary format as required by GCTA. A version showing estimated GRM in the computer program PLINK is also provided.

## 2 A toy example

### 2.1 Data

The data is on a single family from the computer program Morgan.

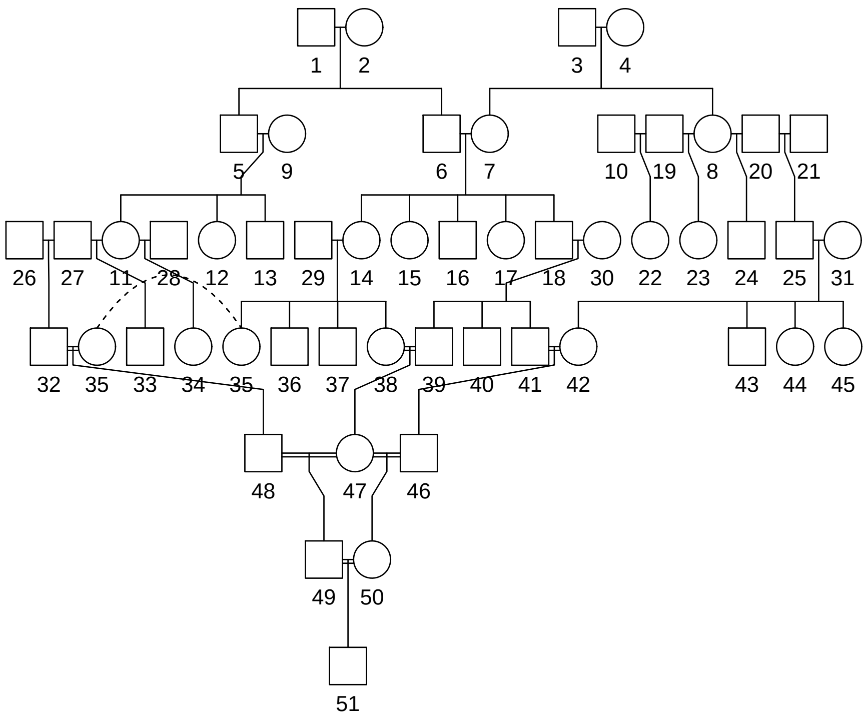
```
> library(gap)
> head(l51,10)

   id fid mid sex aff      qt
1   1   0   0   1   1 -0.9642
2   2   0   0   2   1  1.0865
3   3   0   0   1   1 -0.5363
```

```
4   4   0   0   2   1  0.4514
5   5   1   2   1   1  0.0538
6   6   1   2   1   1 -1.2667
7   7   3   4   2   1       NA
8   8   3   4   2   1  0.1743
9   9   0   0   2   1  0.2923
10 10   0   0   1   1       NA
```

```
> library(kinship2)
> ped <- with(l51,pedigree(id,fid,mid,sex))
> pdf("figures/l51.pdf")
> plot(ped)
> dev.off()
```

```
null device
          1
```

and the pedigree diagram is as follows,



## 2.2 Model

We can obtain a linear mixed model for the quantitative trait (qt) in `l51` above.

```
> library(gap)
> k2 <- kin.morgan(l51)$kin.matrix*2
> k2[1:10,1:10]
```

```
     1   2   3   4   5   6   7   8 9 10
1  1.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0  0
2  0.0 1.0 0.0 0.0 0.5 0.5 0.0 0.0 0  0
3  0.0 0.0 1.0 0.0 0.0 0.0 0.5 0.5 0  0
4  0.0 0.0 0.0 1.0 0.0 0.0 0.5 0.5 0  0
5  0.5 0.5 0.0 0.0 1.0 0.5 0.0 0.0 0  0
6  0.5 0.5 0.0 0.0 0.5 1.0 0.0 0.0 0  0
7  0.0 0.0 0.5 0.5 0.0 0.0 1.0 0.5 0  0
8  0.0 0.0 0.5 0.5 0.0 0.0 0.5 1.0 0  0
9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1  0
10 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0  1
```

```
> library(regress)
> r <- regress(qt ~ 1, ~k2, data=l51)
> r$sigma
```

```
       k2        In
0.2817099 0.4444962
```

```
> r$sigma.cov
```

```
            k2          In
k2  0.07163300 -0.03991478
In -0.03991478  0.04042731
```

The function `kin.morgan` is readily used for the well-ordered pedigree. The relationship matrix is supplied to `regress` function for parameter estimation. We can also generate a binary trait (bt) and run through the regression model similarly,

```
> N <- dim(l51)[1]
> w <- with(l51,quantile(qt,probs=0.75,na.rm=TRUE))
> l51 <- within(l51, bt <- ifelse(qt<=w,0,1))
> with(l51,table(bt))
```

```
bt
 0  1
32 11
```

```
> d <- regress(bt ~ 1, ~k2, data=l51)
> d$sigma
```

```
       k2        In
0.0307703 0.1678370

> d$sigma.cov

             k2           In
k2  0.003615481 -0.002525622
In -0.002525622  0.003492826
```

## 2.3 Heritabilities

Once the mixed models are obtained, we can get the heritability estimates. Note that although we set a population prevalence (K) to be 0.25, there were 11 cases and 40 controls from the simulation, leading to a case/control proportion (P) of 11/51=0.2156863.

The heritability estimate is a ratio of polygenic and phenotypic variance and available from function `h2G` which also gives the associate variance estimate. Internally, this involves function `VR` for calculating variance of a ratio. We illustrate with the example given above,

```
> library(gap)
> # qt
> sigma <- c(0.2817099, 0.4444962)
> sigma.cov <- matrix(
+ c(0.07163300, -0.03991478,
+ -0.03991478, 0.04042731), 2, 2)
> h2G(sigma,sigma.cov)

Vp = 0.7262061 SE = 0.1795292
h2G = 0.38792 SE = 0.3136308

> # bt
> sigma <- c(0.0307703, 0.1678370)
> sigma.cov <- matrix(
+ c(0.003615481, -0.002525622,
+  -0.002525622,  0.003492826), 2, 2)
> h2G(sigma,sigma.cov)

Vp = 0.1986073 SE = 0.04535486
h2G = 0.1549304 SE = 0.2904298
```

As only a single family is involved in the analysis, it is not surprising to see large standard errors. For a case-control study, the heritability estimation is based on a liability threshold model and the connection is furnished through the function `h2l` taking into account the population prevalence and the proportion of cases in the sample (Lee et al. (2011)).

```
> h2l(K=0.25, P=11/51, h2=0.1549304, se=0.2904298)

K =  0.25 P =  0.2156863
h2 = 0.1549304 SE = 0.2904298 h2l = 0.3188476 SE = 0.597706
```

which yields a larger point estimate nevertheless with larger standard error. The relationship between population prevalence and heritability will be seen more clearly later.

It makes sense to illustrate with real data. Before doing that, we would like to indicate that when a model includes gene-environment interaction, (restricted) maximum likelihood estimateors would involve three variance components, heritabilities associated with both polygenic and interaction are obtained via function `h2GE`.

Below is an example from a real session of GCTA analysis but we only keep the variance components and their (lower-triangular) variance-covariance matrix as input to the relevant functions described above.

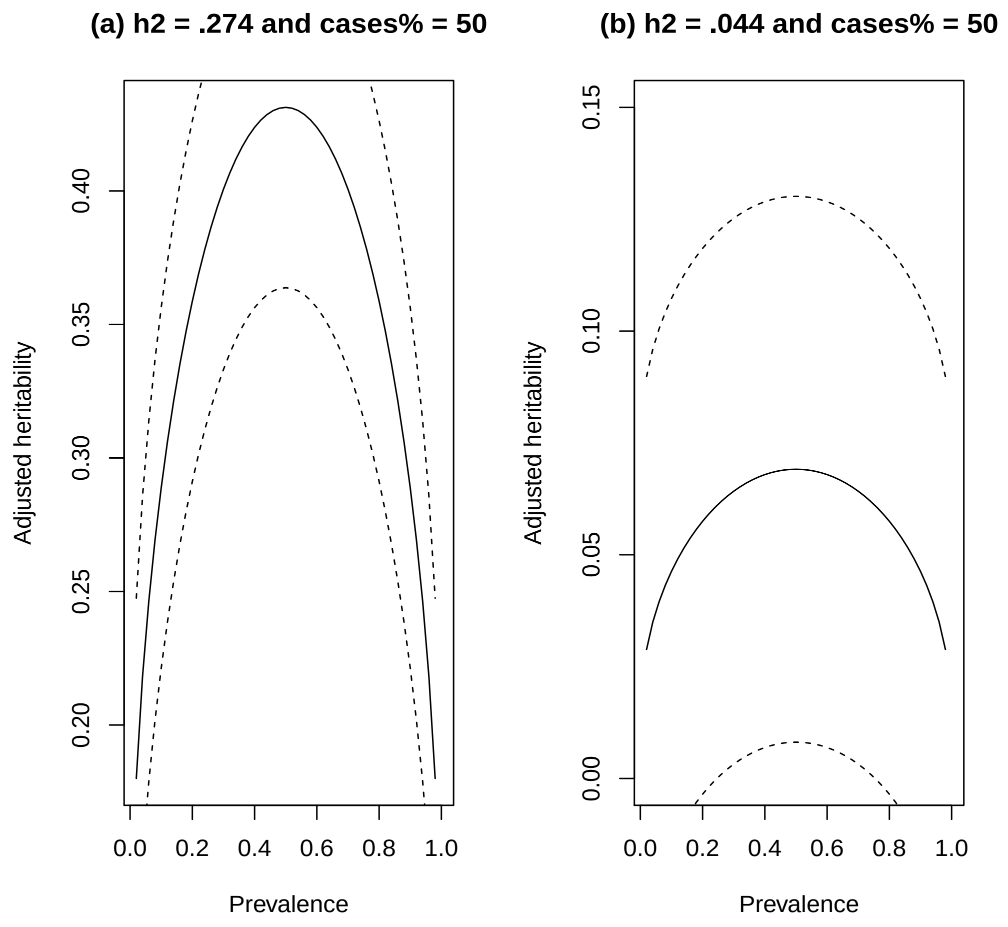
```
> library(gap)
> V <- c(0.017974, 0.002451, 0.198894)
> VCOV <- matrix(0,3,3)
> diag(VCOV) <- c(0.003988, 0.005247, 0.005764)^2
> VCOV[2,1] <- -7.93348e-06
> VCOV[3,1] <- -5.54006e-06
> VCOV[3,2] <- -1.95297e-05
> z <- h2GE(V,VCOV)

Vp = 0.219319 SE = 0.003263797
h2G = 0.08195368 SE = 0.01799574 h2GE = 0.0111755 SE = 0.02392398
```

## 3 Oberved vs scaled heritabilities

Here we explore the relationship between observed and scaled heritability estimates based on a case-control analysis,

```
> library(gap)
> P <- 0.496404
> R <- 50
> kk <- h2all <- seall <- h2alls <- sealls <- rep(0,R)
> for(i in 1:R)
+ {
+   kk[i] <- i/R
+   h2 <- 0.274553
+   se <- 0.067531
+   z <- h2l(kk[i],P=P,h2=h2,se=se,verbose=FALSE)
+   h2all[i] <- z$h2l
+   seall[i] <- z$se
+   h2 <- 0.044
+   se <- 0.061
+   z <- h2l(kk[i],P=P,h2=h2,se=se,verbose=FALSE)
+   h2alls[i] <- z$h2l
+   sealls[i] <- z$se
+ }
> pdf("figures/h2l.pdf")
> par(mfrow=c(1,2))
> plot(kk,h2all,type="l",ylab="Adjusted heritability",xlab="Prevalence")
> lines(kk,h2all-seall,lty="dashed")
> lines(kk,h2all+seall,lty="dashed")
> title("(a) h2 = .274 and cases% = 50")
> plot(kk,h2alls,type="l",ylab="Adjusted heritability",xlab="Prevalence",ylim=c(0,0.15))
> lines(kk,h2alls-sealls,lty="dashed")
> lines(kk,h2alls+sealls,lty="dashed")
> title("(b) h2 = .044 and cases% = 50")
> dev.off()

null device
          1
```

where we set disease prevalence over a grid of 50, as shown in the following figure,



This suggests a nonlinear relationship between the observed and adjusted estimtes and as a function of prevalence.

## 4 Exchange of GRMs between software

We can read or write the GRMs used by GCTA for the example above with the following code,

```
> p <- matrix(0,N,4)
> for(i in 1:N) p[i,] <- with(l51[i,],c(i,i,qt,bt))
> write(t(p),file="51.txt",4,sep="\t")
> NN <- rep(51, N * (N + 1)/2)
> WriteGRM(51,p[,1:2],NN,k2)
> one <- ReadGRM(51)
> grm <- one$grm
> WriteGRMBin(51,grm,NN,p[,1:2])
> two <- ReadGRMBin(51,TRUE)
> sum(one$GRM-two$GRM)
```

As well as illustrating how to manipulate GRMs in two formats, we also generate a phenotypic file called `51.txt`. Note the function `kin.morgan` result has an elemenet called `kin` which is similar to the vector `grm` above.

GRM from PLINK, i.e., the .genome file, can be read via a function called `ReadGRMPLINK`. On reviewing earlier work in relation to package kinship, a simpler implementation is possible esp. with integer ID's with the `bdsmatrix.ibd` function in package `bdsmatrix`, therefore it is added to `gap`'s suggested package list.

Another function is called `WriteGRMSAS` can be used to output an `ldata` as required by type=LIN(1) in SAS PROC MIXED and PROC GLIMMIX. As for phenotypic data, we again turn to our pedigree `l51` and issue commands,

```
> library(foreign)
> write.dta(l51, "l51.dta")
```

to save the data as an external file in Stata format so that software system such as SAS can read it directly. Together with relationship matrix we can take a whole range of facilities available from there. Of course with this particular example, one could use PROC INBREED to generate a relationship matrix.

Morgan actually provides the relevant result for this pedigree as well. It is possible to work on kinship matrix generated from SOLAR, Earlier we discussed how to do this kind of analysis using SAS in Zhao and Luan (2012).

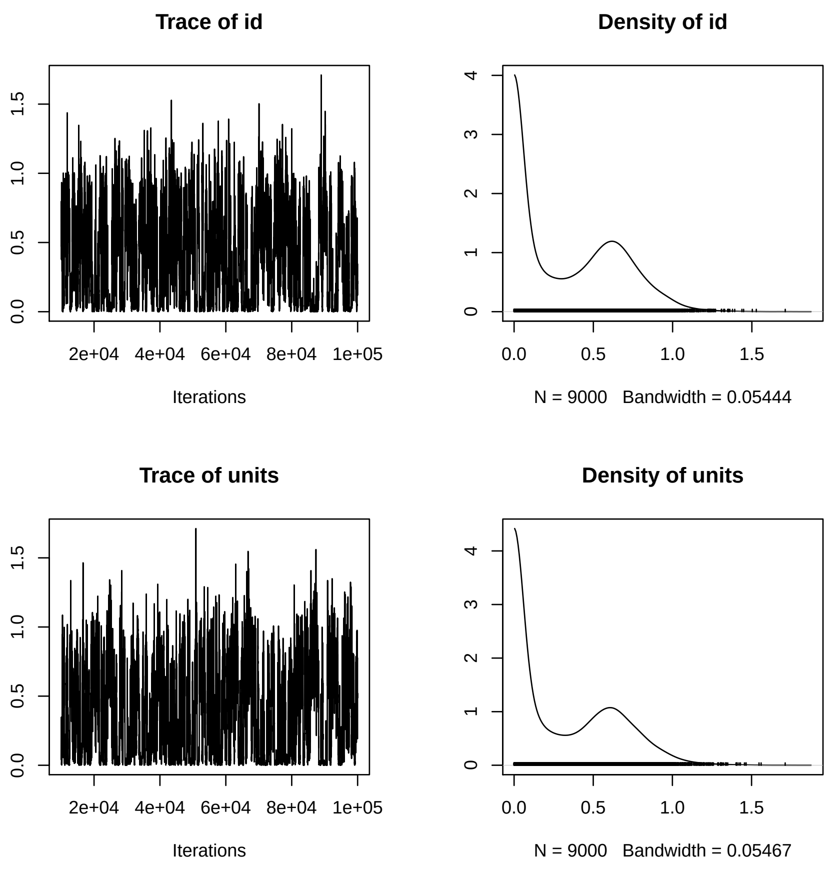
## 5 Inference based on Markov chain Monte Carlo (MCMC)

### 5.1 The toy data

The exmaple also prompts us to seek alternative strategies for inference. Fortunately, we have been able to do so with available facilities in R as detailed below,

First we can take advantage of the family structure,

```
> library(gap)
> library(MCMCglmm)
> prior<-list(R=list(V=1, nu=0.002), G=list(G1=list(V=1, nu=0.002)))
> m <- MCMCglmm(qt~sex,random=~id,data=l51,prior=prior,burnin=10000,nitt=100000,verbose=FALSE)
> summary(m)

 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: 15.57164

 G-structure:  ~id

   post.mean l-95% CI u-95% CI eff.samp
id    0.3812 0.000213   0.8978    244.7

 R-structure:  ~units

      post.mean  l-95% CI u-95% CI eff.samp
units    0.3565 0.0002866   0.8904    236.2

 Location effects: qt ~ sex

            post.mean l-95% CI u-95% CI eff.samp  pMCMC
(Intercept)   0.74745 -0.05619  1.53210     5830 0.0636 .
sex          -0.29457 -0.79481  0.22519     6623 0.2498
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> pdf("figures/MCMCglmm1.pdf")
> plot(m)
> dev.off()

null device
          1
```



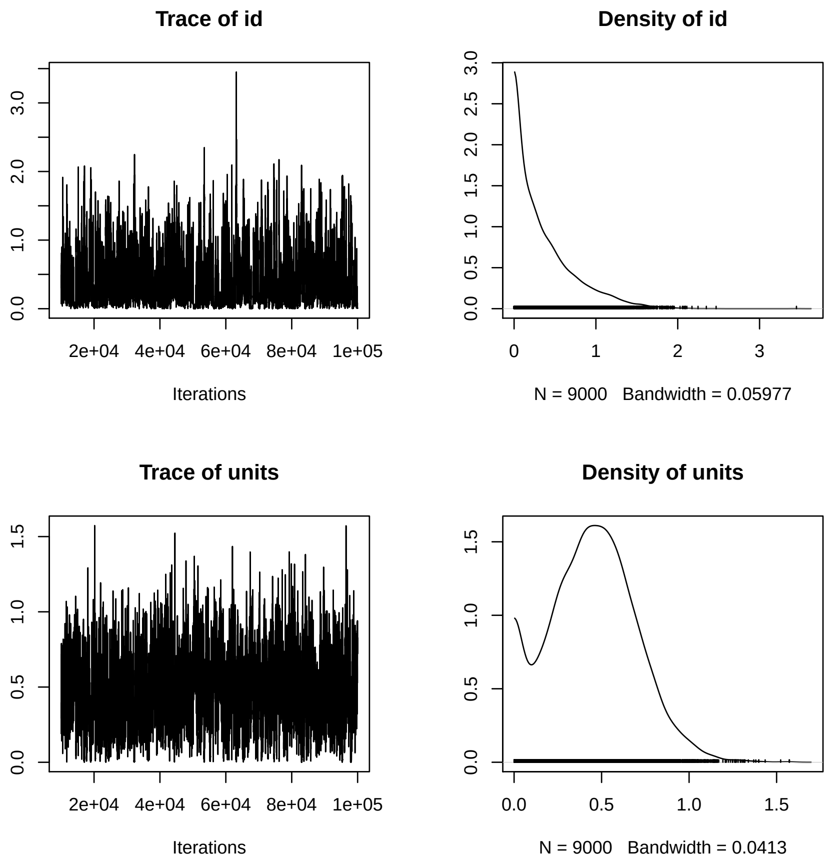
We next seek to use the kinship matrix directly.

```
> library(gap)
> km <- kin.morgan(l51)
> k2 <- km$kin.matrix*2
> N <- 51
> i <- rep(1:N,rep(N,N))
> j <- rep(1:N,N)
> library(Matrix)
> s <-spMatrix(N,N,i,j,as.vector(k2))
> Ginv<-solve(s)
> class(Ginv) <- "dgCMatrix"
> rownames(Ginv) <- Ginv@Dimnames[[1]] <- with(l51,id)
> library(MCMCglmm)
> prior<-list(R=list(V=1, nu=0.002), G=list(G1=list(V=1, nu=0.002)))
> m <- MCMCglmm(qt~1, random=~id, ginverse=list(id=Ginv), data=l51, prior=prior,
+               burnin=10000, nitt=100000, verbose=FALSE)
> summary(m)
> save(m,file="MCMCglmm.fit")
> pdf("MCMCglmm2.pdf")
> plot(m$VCV)
> dev.off()
```

It ran fairely fast on the Linux system and the summary statistics are as follows,

```
> summary(m)

 Iterations = 10001:99991
 Thinning interval  = 10
 Sample size  = 9000

 DIC: 82.43772

 G-structure:  ~id

   post.mean  l-95% CI u-95% CI eff.samp
id    0.3811 0.0003015    1.166    620.9

 R-structure:  ~units

      post.mean l-95% CI u-95% CI eff.samp
units    0.4564 0.000302    0.855    817.7

 Location effects: qt ~ 1

            post.mean l-95% CI u-95% CI eff.samp  pMCMC
(Intercept)  0.469000 0.004706 1.028013     2207 0.0364 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The convergence plots are shown as the following graph,



In the code, we also saved the model for examination. This example actually led to addition of packages `Matrix` and `MCMCglmm` (Hadfield (2010)) to `gap`'s suggested package list. As the procedure is fairly general, it is worthwhile and much simpler to wrap up as a dedicated function (`MCMCgrm`) in which case the call becomes,

```
> s <- kin.morgan(l51)
> K <- with(s,kin.matrix*2)
> prior <- list(R=list(V=1, nu=0.002), G=list(G1=list(V=1, nu=0.002)))
> m <- MCMCgrm(qt~1,prior,l51,K,n.burnin=10000, n.iter=100000)
> save(m,file="l51.m")
> plot(m)
```

Interestingly, inside the function `solve` needs to have a scope operator, i.e., `Matrix::solve`, to enable `Ginv` to be S4 object. This is a nuisance but not a big overhead.

### 5.2 A simulated data

The next example is according to Meyer (1989); Tempelman and Rosa (2004) on 282 animals from 24 populations, from which we obtain the restricted maximum likelihood (REML) estimates first, to be followed by two versions of MCMC.

```
> meyer <- within(meyer,{
+    g1 <- ifelse(generation==1,1,0)
+    g2 <- ifelse(generation==2,1,0)
+ })
> # library(kinship)
> # A <- with(meyer,kinship(animal,sire,dam))*2
> # Here we convert NAs to 0s to be compatible with kin.morgan
> meyer0 <- within(meyer,{
+    id <- animal
+    animal <- ifelse(!is.na(animal),animal,0)
+    dam <- ifelse(!is.na(dam),dam,0)
+    sire <- ifelse(!is.na(sire),sire,0)
+    g1 <- ifelse(generation==1,1,0)
+    g2 <- ifelse(generation==2,1,0)
+ })
> A <- kin.morgan(meyer0)$kin.matrix*2
> library(regress)
> r <- regress(y~-1+g1+g2,~A,data=meyer0)
> summary(r)

Likelihood kernel: K = g1+g2

Maximized log likelihood with kernel K is  -754.555

Linear Coefficients:
    Estimate Std. Error
 g1  220.321      1.725
 g2  236.695      1.999

Variance Coefficients:
    Estimate Std. Error
 A    43.955     16.998
 In   50.953     10.594

> with(r,h2G(sigma,sigma.cov))

Vp = 94.9083 SE = 10.58581
h2G = 0.4631322 SE = 0.1410644

> library(MCMCglmm)
> m <-MCMCglmm(y~-1+g1+g2,random=animal~1,pedigree=meyer[,1:3],data=meyer,verbose=FALSE)
> summary(m)
```

```
 Iterations = 3001:12991
 Thinning interval  = 10
 Sample size  = 1000

 DIC: 2000.055

 G-structure:  ~animal

       post.mean l-95% CI u-95% CI eff.samp
animal      45.6     21.6    70.59    349.5

 R-structure:  ~units

      post.mean l-95% CI u-95% CI eff.samp
units     50.71    35.69    67.27    408.3

 Location effects: y ~ -1 + g1 + g2

   post.mean l-95% CI u-95% CI eff.samp  pMCMC
g1     220.4    216.7    223.9     1000 <0.001 ***
g2     236.8    232.7    240.4     1000 <0.001 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> plot(m)
> prior <- list(R=list(V=1, nu=0.002), G=list(G1=list(V=1, nu=0.002)))
> m2 <- MCMCgrm(y~-1+g1+g2,prior,meyer0,A,singular.ok=TRUE,verbose=FALSE)
> summary(m2)

 Iterations = 3001:12991
 Thinning interval  = 10
 Sample size  = 1000

 DIC: 1999.658

 G-structure:  ~id

   post.mean l-95% CI u-95% CI eff.samp
id     46.24    20.18    72.12    332.8

 R-structure:  ~units

      post.mean l-95% CI u-95% CI eff.samp
units     50.87    34.91    69.18    544.7

 Location effects: y ~ -1 + g1 + g2
```

```
   post.mean l-95% CI u-95% CI eff.samp  pMCMC
g1     220.3    217.1    223.8     1137 <0.001 ***
g2     236.7    232.8    240.5     1282 <0.001 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> plot(m2)
```

The MCMC procedures use pedigree structures or genetic relationships, respectively. It is good to have narrawer confidence intervals for the variance components from these.

## 5.3 Using OpenBUGS and JAGS

It is handy to use `MCMCgrm` as described above, but there are two aspects which we would like to explore. I found it still very slow with moderate sample size. Instead of ploughing into the implementation could we use the simpler and familiar syntax in OpenBUGS and JAGS? and if that is the case we can resort to faster setup such as Stan (`http://mc-stan.org/`).

We start with OpenBUGS and for illustrative purpose we continue to use the toy data in section 5.1 but impute the missing data for variable `qt`.

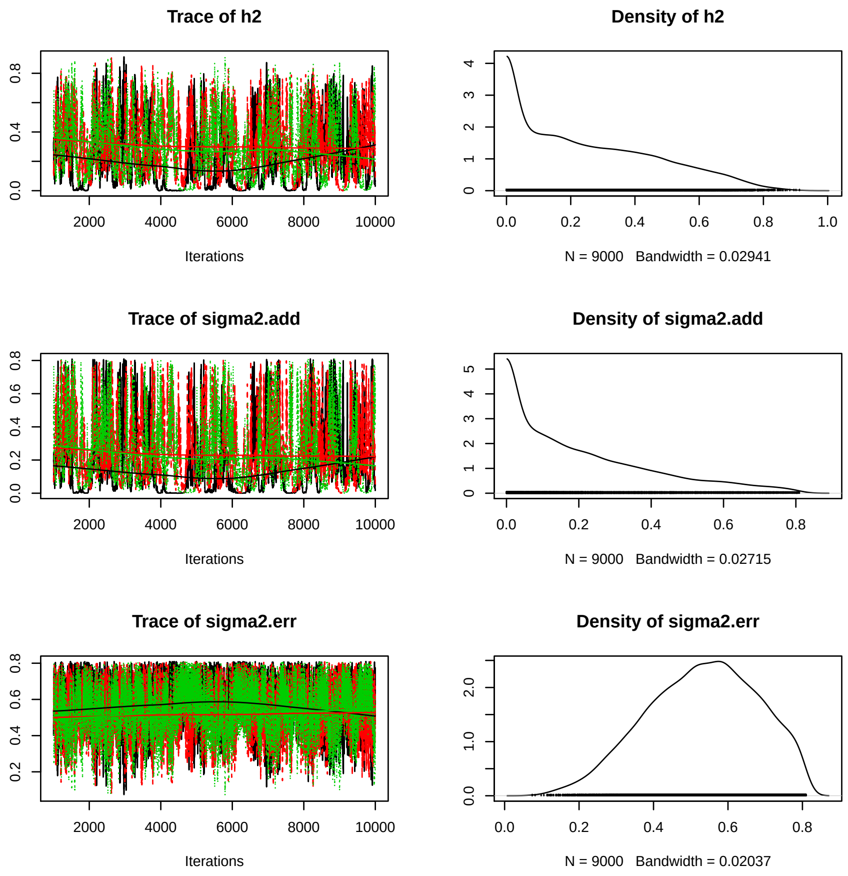
```
> library(gap)
> set.seed(1234567)
> ped51 <-
+ within(l51, {qt[is.na(qt)] <- rnorm(length(qt[is.na(qt)]),
+              mean(qt,na.rm=TRUE),sd(qt,na.rm=TRUE));})
> l51 <- rbind(subset(ped51,fid==0),subset(ped51,fid!=0))
> data=with(l51,list(n=51,f=15,f1=16,m=2,Y=qt,X=sex,FID=fid,MID=mid,
+               sd.u.add=0.9,sd.u.err=0.9))
> inits=function()list(beta=c(0,0))
> library(R2OpenBUGS)
> bugs.data(data,data.file="data.txt")

[1] "data.txt"

> bugs.inits(inits,n.chains=3,digits=3)

[1] "inits1.txt" "inits2.txt" "inits3.txt"

> bugsfit <- bugs(data,
+                 inits,
+                 parameters.to.save=c("beta","sigma2.add","sigma2.err","h2"),
+                 model.file="model.txt",
+                 n.chains=3,
+                 n.burnin=1000,
+                 n.iter=10000,
+                 codaPkg=TRUE)
> library(coda)
> pdf("figures/bugs.pdf")
> bugsfit.coda <- read.bugs(bugsfit)

Abstracting beta[1] ... 9000 valid values
Abstracting beta[2] ... 9000 valid values
Abstracting deviance ... 9000 valid values
Abstracting h2 ... 9000 valid values
Abstracting sigma2.add ... 9000 valid values
Abstracting sigma2.err ... 9000 valid values
Abstracting beta[1] ... 9000 valid values
Abstracting beta[2] ... 9000 valid values
Abstracting deviance ... 9000 valid values
Abstracting h2 ... 9000 valid values
Abstracting sigma2.add ... 9000 valid values
Abstracting sigma2.err ... 9000 valid values
Abstracting beta[1] ... 9000 valid values
Abstracting beta[2] ... 9000 valid values
Abstracting deviance ... 9000 valid values
Abstracting h2 ... 9000 valid values
Abstracting sigma2.add ... 9000 valid values
Abstracting sigma2.err ... 9000 valid values

> summary(bugsfit.coda)

Iterations = 1001:10000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 9000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                Mean      SD  Naive SE Time-series SE
beta[1]      0.55684  0.2193 0.0013345       0.004216
beta[2]      0.02853  0.2207 0.0013431       0.003994
deviance   110.63930 14.9326 0.0908771       0.619530
h2           0.27264  0.2135 0.0012993       0.010957
sigma2.add   0.22088  0.1971 0.0011996       0.009824
sigma2.err   0.53045  0.1479 0.0008999       0.004620

2. Quantiles for each variable:

                2.5%       25%      50%      75%    97.5%
beta[1]     0.127298   0.41280   0.5557   0.6982   0.9996
beta[2]    -0.407305  -0.11770   0.0295   0.1734   0.4658
deviance   72.629250 103.20000 114.4000 122.1000 128.4000
h2          0.001025   0.08623   0.2360   0.4295   0.7219
```

```
sigma2.add  0.000661   0.05817   0.1681   0.3385   0.7013
sigma2.err  0.231192   0.42528   0.5383   0.6429   0.7835

> plot(bugsfit.coda)
> dev.off()

pdf
  2
```

where `model.txt` is taken directly from Waldmann (2009), which requires founders precede the nonfounders and that is the reason we have the statement containing two subset commands.

```
model
{
  # Loop over all individuals for inference of error precision
  for(i in 1 : n) {
    Y[i] ~ dnorm(mu[i], tau.err)
  }
  # Loop over founders for inference of additive values and precision
  for (i in 1 : f){
    mu[i] <- mean(beta[]) + add[i]
    add[i] ~ dnorm(0, tau.add)
  }
  # Loop over descendants for inference of additive values and precision
  for (i in f1 : n){
    mu[i] <- beta[X[i]] + add[i]
    par.add[i] <- (add[FID[i]] + add[MID[i]])/2.0
    add[i] ~ dnorm(par.add[i], prec.add)
  }
  # Specification of prior distributions
  for (i in 1: m){
    beta[i] ~ dnorm(0.0, 1.0E-6)
  }
  tau.add <- 1 / sigma2.add
  sigma.add ~ dunif(0, sd.u.add)
  sigma2.add <- sigma.add * sigma.add
  tau.err <- 1 / sigma2.err
  sigma.err ~ dunif(0, sd.u.err)
  sigma2.err <- sigma.err * sigma.err
  prec.add <- 2 * tau.add
  # Specification of functions of model parameters of inferential interest
  h2 <- sigma2.add / (sigma2.add + sigma2.err)
}
```

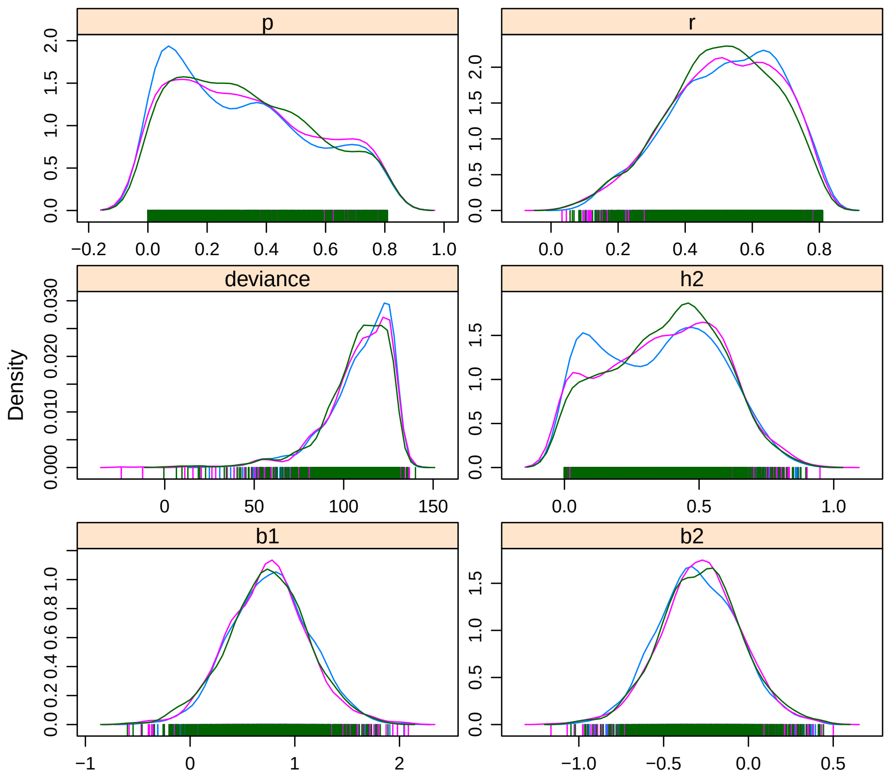
To streamline the results, we resort to package `coda`. In the coding we also output data and initial values which allow for use in OpenBUGS itself.



where we skip figures for the fixed effects and deviance while choosing to show the variance components only.

As before, we would still be keen to use GRM rather than a pedigree structure. We alter the coding above slightly and use JAGS instead, As before, we take advantage of the facility in package `regress` for the familiar REML estimation.

```
> library(gap)
> set.seed(1234567)
> km <- kin.morgan(l51)
> k2 <- km$kin.matrix*2
> l51 <-
+ within(l51, {qt[is.na(qt)] <- rnorm(length(qt[is.na(qt)]),
+               mean(qt,na.rm=TRUE),sd(qt,na.rm=TRUE))})
> N <- dim(l51)[1]
> data=with(l51,list(N=N,qt=qt,sex=sex,GI=solve(k2),u=rep(0,N)))
> library(regress)
> r <- regress(qt ~ sex, ~k2, data=data)
> r

Likelihood kernel: K = (Intercept)+sex

Maximized log likelihood with kernel K is  -14.253

Linear Coefficients:
             Estimate Std. Error
 (Intercept)    0.793      0.367
 sex           -0.300      0.230

Variance Coefficients:
             Estimate Std. Error
          k2    0.290      0.236
          In    0.435      0.177

> with(r,{
+   print(sqrt(sigma+1.96*sqrt(diag(sigma.cov))))
+   h2G(sigma,sigma.cov)
+ })

      k2       In
0.867537 0.884864
Vp = 0.7254864 SE = 0.1643443
h2G = 0.4001416 SE = 0.2739208

> inits=function()list(b1=0,b2=0,sigma.p=0.001,sigma.r=0.001)
> modelfile=function() {
+     b1 ~ dnorm(0, 0.001)
+     b2 ~ dnorm(0, 0.001)
+     sigma.p ~ dunif(0,0.9)
+     sigma.r ~ dunif(0,0.9)
+     p <- pow(sigma.p, 2)
+     r <- pow(sigma.r, 2)
+     h2 <- p / (p + r)
+     tau <- pow(sigma.r, -2)
+     xi ~ dnorm(0,tau.xi)
+     tau.xi <- pow(0.9,-2)
+     g[1:N] ~ dmnorm(u[],GI[,]/p)
+     for(i in 1:N) {qt[i] ~ dnorm(b1 + b2 * sex[i] + xi*g[i],tau)}
+ }
> library(R2jags)
> jagsfit <- jags(data,
+                 inits,
+                 parameters.to.save=c("b1","b2","p","r","h2"),
+                 model.file=modelfile,
+                 n.chains=3,
+                 n.burnin=1000,
+                 n.iter=10000)

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 2930

Initializing model

> save(jagsfit,file="jags.fit")
> print(jagsfit)

Inference for Bugs model at "/tmp/RtmpT7Qb5g/model565b199c4eb.txt", fit using jags,
 3 chains, each with 10000 iterations (first 1000 discarded), n.thin = 9
 n.sims = 3000 iterations saved
         mu.vect sd.vect   2.5%    25%     50%     75%   97.5%  Rhat n.eff
b1         0.751   0.381  0.004  0.498   0.755   1.000   1.512 1.002  1600
b2        -0.284   0.237 -0.745 -0.440  -0.286  -0.127   0.193 1.002  1400
h2         0.362   0.212  0.003  0.189   0.378   0.527   0.746 1.021  3000
p          0.328   0.233  0.002  0.126   0.298   0.500   0.782 1.016  3000
r          0.516   0.160  0.180  0.407   0.527   0.642   0.779 1.002  1300
deviance 108.201  18.568 58.132 99.936 111.723 122.093 129.673 1.002  1200

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 172.2 and DIC = 280.4
DIC is an estimate of expected predictive error (lower deviance is better).

> pdf("figures/jags.pdf")
> plot(jagsfit)
> library(lattice)
> jagsfit.mcmc <- as.mcmc(jagsfit)
> traceplot(jagsfit.mcmc)
> xyplot(jagsfit.mcmc)
> densityplot(jagsfit.mcmc)
> dev.off()

pdf
  2
```



We see that the results are largely comparable though it appears that the BUGS version deviates somewhat more from `MCMCgrm`. We are hopeful though to get the idea through to Stan but we omit that for now.

## 5.4 Uncertainty in heritability estimation

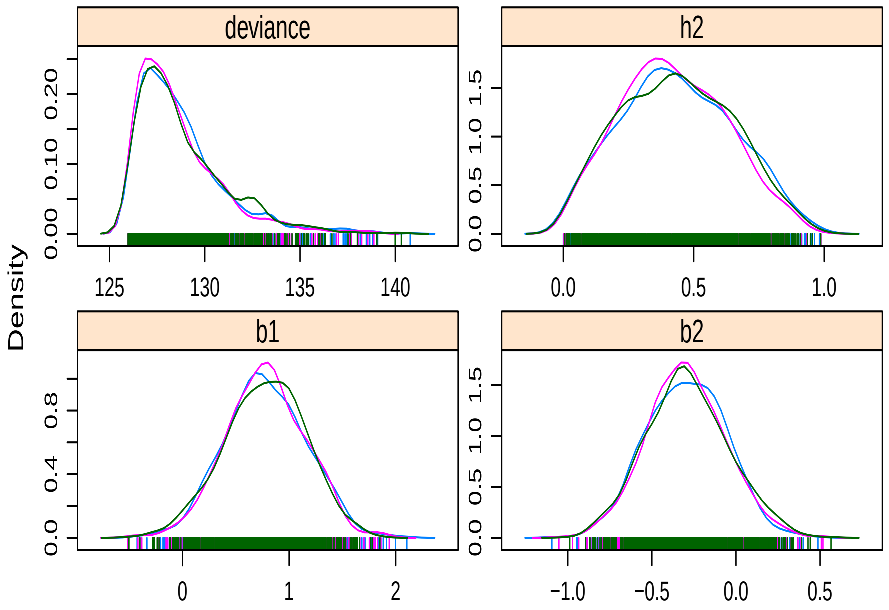
By now we have probably seen enough of MCMC for the toy data, but there is still one more point to give.

The results shown in this section follow closely to Furloette et al. (2014), which is quite similar to the implementation above but uses multivariate t distribution instead. Again we use the toy data from 5.1, and the `R` and JAGS code are as follows, We can see that it is straightforward to take the kinship and identity matrices into JAGS, leading to a greatly simplified program.

```
> library(gap)
> set.seed(1234567)
> ped51 <-
+ within(l51, {
```

```
+   qt[is.na(qt)] <- rnorm(length(qt[is.na(qt)]),mean(qt,na.rm=TRUE),sd(qt,na.rm=TRUE))
+ })
> km <- kin.morgan(l51)
> k2 <- km$kin.matrix*2
> N <- dim(l51)[1]
> data=with(ped51,list(N=N,qt=qt,sex=sex,G=k2,I=diag(N),alpha=1,gamma=1))
> inits=function()list(b1=0,b2=0,h2=0.4)
> modelfile=function() {
+     h2 ~ dunif(0,1)
+     Omega <- inverse((h2*G[,] + (1-h2)*I[,])*gamma/alpha)
+     qt[1:N] ~ dmt(mu[],Omega[,],2*alpha)
+     mu[1:N] <- b1 + b2 * sex[]
+     b1 ~ dnorm(0, 0.001)
+     b2 ~ dnorm(0, 0.001)
+ }
> library(R2jags)
> jagsfit <- jags(data,inits,parameters.to.save=c("b1","b2","h2"),
+                 model.file=modelfile, n.chains=3, n.burnin=1000, n.iter=10000)

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 5277

Initializing model

> save(jagsfit,file="h2.fit")
> print(jagsfit)

Inference for Bugs model at "/tmp/RtmpT7Qb5g/model565b19d81729.txt", fit using jags,
 3 chains, each with 10000 iterations (first 1000 discarded), n.thin = 9
 n.sims = 3000 iterations saved
         mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
b1         0.779   0.380   0.035   0.524   0.778   1.033   1.488 1.001  3000
b2        -0.293   0.239  -0.760  -0.460  -0.300  -0.134   0.191 1.001  3000
h2         0.431   0.211   0.055   0.271   0.422   0.587   0.844 1.001  3000
deviance 128.734   2.280 126.132 127.077 128.135 129.734 134.846 1.002  1200

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 2.6 and DIC = 131.3
DIC is an estimate of expected predictive error (lower deviance is better).

> pdf("figures/h2.pdf")
> plot(jagsfit)
> library(lattice)
> jagsfit.mcmc <- as.mcmc(jagsfit)
> traceplot(jagsfit.mcmc)
> xyplot(jagsfit.mcmc)
> densityplot(jagsfit.mcmc)
> dev.off()

pdf
  2
```

with sigma and alpha both setting to be one, the $h^2$ takes the position of variance compoenent for polygeneic effects. We now have



i.e., the distribution with respect to a single variable $h^2$. This serves as a good correspondence to what we have seen in section 3. A major difference betwee results from pedigree structure and kinship matrix is with respect to the distribution of $h^2$, due to the similarity in the two JAGS implementations their greater agreement is perhaps not surprising.

## 6 Acknowledgements

I wish to thank authors of software `Morgan`, `MCMCglmm` and `GCTA` which made this documentation possible.

## References

A.G. Day-Williams, J. Blangero, T.D. Dyer, K. Lange, and E.M. Sobel. Linkage analysis without defined pedigrees. *Genet Epidemiol.*, 35(5):360–370, July 2011. URL `PM:21465549`.

N. A. Furloette, D. Heckerman, and C. Lippert. Quantifying the uncertainty in heritability. *J Hum Genet*, 59:269–275, 2014.

J.D. Hadfield. MCMC methods for multi-response generalized linear mixed models: The MCMCglmm R package. *J Stat Soft*, 33(2):1–22, 2010. URL `http://www.jstatsoft.org/v33/i02/`.

Y.C. Klimentidis, A.I. Vazquez, Campos G. de Los, D.B. Allison, M.T. Dransfield, and V.J. Thannickal. Heritability of pulmonary function estimated from pedigree and whole-genome markers. *Front Genet*, 4(174):1–5, 2013. URL `PM:24058366`.

S.H. Lee, N.R. Wray, M.E. Goddard, and P.M. Visscher. Estimating missing heritability for disease from genome-wide association studies. *Am J Hum Genet*, 88(3):294–305, March 2011. URL `PM:21376301`.

K. Meyer. Restricted maximum likelihood to estimate variance components for animal models with several random effects using a derivative-free algorithm. *Genet Sel Evol*, 21:317 – 340, 1989.

R. J. Tempelman and G. J. M. Rosa. Empirical bayes approaches to mixed model inference in quantitative genetics. In A. M. Saxton, editor, *Genetic Analysis of Complex Traits Using SAS*, pages 149–176. SAS Institute Inc., Cary, NC, 2004.

P. Waldmann. Easy and flexible bayesian inference of quantitative genetic parameter. *Evolution*, doi:10.1111/j.1558-5646.2009.00645.x:1640–1643, 2009.

J. Yang, B. Benyamin, B.P. McEvoy, S. Gordon, A.K. Henders, D.R. Nyholt, P.A. Madden, A.C. Heath, N.G. Martin, G.W. Montgomery, M.E. Goddard, and P.M. Visscher. Common snps explain a large proportion of the heritability for human height. *Nat Genet*, 42(7):565–569, July 2010. URL `PM:20562875`.

J. Yang, S.H. Lee, M.E. Goddard, and P.M. Visscher. GCTA: A tool for genome-wide complex trait analysis. *Am.J.Hum.Genet*, 88(1):76–82, January 2011. URL `PM:21167468`.

J.H. Zhao and J.A. Luan. Mixed modeling with whole genome data. *J Prob Stat*, doi 10.1155/2012.485174:1–16, 2012. URL `http://www.hindawi.com/journals/jps/2012/485174/`.

.