GWAS on your notebook:
Semi-parallel linear and logistic regression

Presented by

Karolina Sikorska

Department of Biostatistics,
Erasmus Medical Centre in Rotterdam

Hosted by
Shawn Yarnes
Plant Breeding and Genomics

GWAS on your notebook

Semi-parallel linear and logistic regression

Karolina Sikorska and Paul Eilers

Erasmus MC, Rotterdam, The Netherlands

September 12, 2013

Motivation

- Analysis of GWAS usually involves computing clusters
- Parallel computing
- Organization of the SNP data not efficient
- Difficult to extract blocks of SNPs

Our goals

- Speed-up computations by using matrix operations (semi-parallel computing)
- Rearrange data structure using matrix oriented binary files
- Make GWA scans feasible on a notebook

PC, software and speed

- Our PC: Intel i5-3470, 3.20 GHz, 8 GB RAM
- R software, 64-bit version 3.0.0
- We measure speed:
  - n individuals, m SNPs
  - t - time to complete the job (proc.time)
  - speed: $v = mn/t$
  - units - sips (SNPs times individual per second)
  - Numbers are big so we use Msips
  - Flexible to recalculate for different n and m

Outline

- Part I: Linear regression
- Part II: Logistic regression
- Part III: Data access

Part I: Linear regression

Linear regression

Simple model (no covariates):

$$y = \alpha + \beta s + \epsilon$$

Most straightforward: function **lm** in a loop

GWA analysis in a loop using lm

```
### Simulate the data
set.seed(2013)
n = 10000
m = 10000
S = matrix(2 * runif(n * m), n, m)
y = rnorm(n)
### Analyze
t0 = proc.time()[1]
beta = rep(NA, m)
for(i in 1 : m) {
  mod1 = lm(y ~ S[ , i])
  beta[i] = mod1$coeff[2]
}
t1 = proc.time()[1] - t0
speed = (m * n)/(t1 * 1e06)
cat(sprintf("Speed: %2.1f Msips\n", speed))
```

Speed: 1 Msips (For 10K individuals and 1M SNPs $\approx$ 3 hours)

GWA in a loop using lsfit

```
beta = rep(NA, m)
for(i in 1 : m) {
  mod1 = lsfit(S[ , i], y)
  beta[i] = mod1$coeff[2]
}
```

Speed: 6.9 Msips

Explicit solution, still loop

Solution for $\widehat{\beta}$:

$$\widehat{\beta} = \frac{\sum_{i=1}^{n}(s_i - \bar{s})(y_i - \bar{y})}{\sum_{i=1}^{n}(s_i - \bar{s})^2}$$

```
beta = rep(NA, m)
yc = y - mean(y)
for(i in 1 : m){
  sc = S[ , i] - mean(S[ , i])
  beta[i] = sum(sc * yc) / sum(sc ^ 2)
}
```

Speed: 45 Msips

Semi-parallel computations

- Note that $\widehat{\beta} = \frac{\sum_{i=1}^{n} \tilde{y}\tilde{s}}{\sum_{i=1}^{n} \tilde{s}^2}$, where $\tilde{s}$ and $\tilde{y}$ are centered s and y
- Take S: block of k SNPs
- Vector of k $\widehat{\beta}$'s computed at once by $y^T \tilde{S}/\text{colSums}(\tilde{S}^2)$
- Semi-parallel regression (SPR)
- Many SNPs analyzed in parallel but on the same computer

SPR in R using scale

```
yc = y - mean(y)
Sc = scale(S, center = TRUE, scale = FALSE)
s2 = colSums(Sc ^ 2)
beta = crossprod(yc, sc)/s2
```

Speed: 32 Msips. Scale function is slow.

SPR avoiding scale

We center SNP matrix ourselves

```
yc = y - mean(y)
s1 = colSums(S)
e = rep(1, n)
Sc = S - outer(e, s1/n)
beta = crossprod(yc, Sc)/colSums(Sc ^ 2)
```

Speed: 130 Msips

More tricks

Centering not necessary.

$$\sum_i^n \tilde{y}_i(s_i - \bar{s}) = \sum_i^n \tilde{y}_i s_i - \bar{s}\sum_i^n \tilde{y}_i = \sum_i^n \tilde{y}_i s_i.$$

$$\sum_i^n (s_i - \bar{s})^2 = \sum_i^n s_i^2 - n(\bar{s})^2$$

```
yc = y - mean(y)
s1 = colSums(S)
s2 = colSums(S ^ 2)
beta = crossprod(yc, S)/(s2 - (s1 ^ 2)/ n)
```

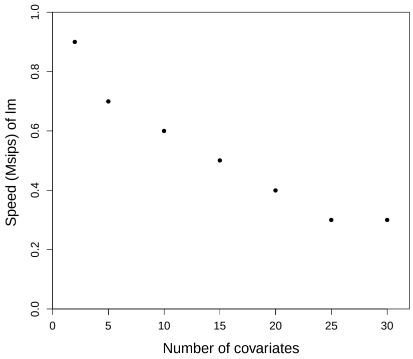
Speed: 220 Msips.
10K individuals, 1M SNPs done within a minute
(compare to 3 hours)

Regression with covariates

Model:

$$y = \beta s + X\gamma + \epsilon$$

- Assume that X includes intercept
- Easily added to lm (or lsfit), but note how it affects the speed



Regression with covariates - projections

If we introduce:

$$s^* = s - X(X^T X)^{-1} X^T s$$

$$y^* = y - X(X^T X)^{-1} X^T y$$

$\widehat{\beta}$ is a solution of new model

$$y^* = \beta s^* + \epsilon$$

Solved by very simple formula

$$\hat{\beta} = \sum_i^n s_i^* y_i^* / \sum_i^n {s_i^*}^2$$

SPR with covariates

```
n = 10000
m = 10000
k = 30
S = matrix(2 * runif(n *m), n , m)
X0 = matrix(rnorm(n * k), n, k)
X = cbind(1, X0)
y = rnorm(n)
### transform y
U1 = crossprod(X, y)
U2 = solve(crossprod(X), U1)
ytr = y - X %*% U2
```

Transform all SNPs at once

```
U3 = crossprod(X, S)
U4 = solve(crossprod(X), U3)
Str = S - X %*% U4
## compute slopes
b = crossprod(ytr, Str)/colSums(Str ^ 2)
```

Standard errors and p-values

Given model

$$y^* = \beta s^* + \epsilon$$

the variance of $\widehat{\beta}$ is estimated by

$$\text{var}(\widehat{\beta}) = \hat{\sigma}^2 ({s^*}^T s^*)^{-1}$$

Errors variance:

$$\hat{\sigma}^2 = \frac{\text{RSS}}{n-k-2}$$

$$\text{RSS} = \sum_i^n {y_i^*}^2 - \widehat{\beta}^2 \sum_i^n {s_i^*}^2$$

Standard errors and p-values in SPR

```
S_sq = colSums(Str ^ 2)
RSS = sum(ytr ^ 2) - b ^ 2 * S_sq
sigma_hat = RSS/(n - k - 2)
error = sqrt(sigma_hat/ S_sq)
pval = 2 * pnorm(-abs (b / error))
```

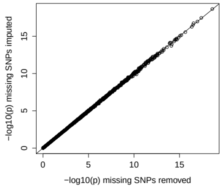
Final speed comparison

k	**lm**	**lsfit**	**SPR**
2	0.9	3.2	50
5	0.7	2.3	45
10	0.6	1.6	40
30	0.3	0.5	20

10K individuals, 1M SNPs, 10 covariates done within 5 minutes.

Missing data

- Missing response not a problem
- Missing SNPs more difficult (however not common with recent imputations)
- SPR does not allow for "NA" in a SNP block
- Our solution: impute missing genotypes with a sample mean
- It works well (example with n = 1000, and missingness rate 5%)



Part II: Logistic regression

Estimation in logistic regression

- Model

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 s$$

- GLM framework, typically fitted with maximum likelihood
- Iterative procedure (Newton-Raphson)
- 4-5 times slower than least squares
- Not possible to (semi-)parallelize
- Our proposal: Write ML as iteratively reweighted least squares

Logistic regression in R

```
S = matrix(2 * runif(n * m), n , m )
y = rbinom(n, size = 1, prob = c(0.5, 0.5))
beta = rep(NA, m)
t0 = proc.time()[1]
for(i in 1 : m){
  mod1 = glm( y ~ S[ , i], family = binomial(link = logit))
  beta[i] = summary(mod1)$coef[2 , 1]
}
t1 = proc.time()[1] - t0
speed = (m * n)/(t1 * 1e06)
cat(sprintf("Speed: %2.1f Msips\n", speed))
```

```
Speed: 0.2 Msips
```

Iteratively reweighted least squares

Write maximum likelihod equation for (t+1)th iteration as:

$$(X^T W^{(t)} X)\beta^{(t+1)} = X^T W^{(t)} z^{(t)},$$

where

$$z_i^{(t)} = \log\left(\frac{p_i^{(t)}}{1-p_i^{(t)}}\right) + \frac{y_i - p_i^{(t)}}{p_i^{(t)}(1-p_i^{(t)})}$$

and $W^{(t)}$ is diagonal matrix with elements $p_i^{(t)}(1-p_i^{(t)})$

$$\text{cov}(\hat{\beta}^{(t+1)}) = (X^T W^{(t)} X)^{-1}$$

Iteratively reweighted least squares (equivalent to glm)

```
ps = rep(mean(y), n)
X = cbind(1, s)
for(i in 1:20) {
  wi = ps * (1 - ps)
  W = diag(wi, n, n)
  zi = log(ps/(1 - ps)) + (y - ps)/wi
  M1 = t(X) %*% W %*% X
  M2 = solve(M1)
  bethat = M2 %*% t(X) %*% W %*% zi
  b0 = bethat[1]
  b1 = bethat[2]
  num1 = exp(b0 + b1 * s)
  ps = num / (1 + num)
}
varhat = sqrt(diag(M2))
```

Semi-parallel logistic regression

- In principle weights are SNP-dependent
- Exact semi-parallel approach cannot be applied
- But SNP effects are very small
- Weights from the model without SNP are almost final
- Treat SNP as perturbation to that model

SP logistic regression without covariates

SNP effect from weighted least squares

$$\widehat{\beta}_1 = \frac{\sum_i w_i(z_i - z_w)(s_i - s_w)}{\sum_i w_i(s_i - sw)^2}$$

$$var(\beta_1) = \frac{1}{\sum_i w_i(s_i - s_w)^2},$$

with $s_w = \sum_i w_i s_i / \sum_i w_i$ and $z_w = \sum_i w_i z_i / \sum_i w_i$

- w_i are the same for all individuals
- Formula for $\widehat{\beta}_1$ same like for linear regression

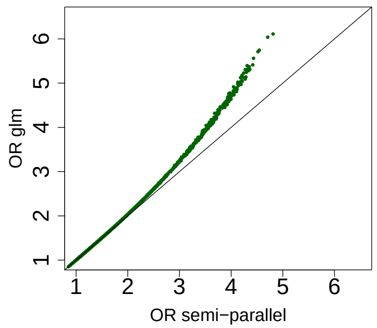
SP logistic regression without covariates in R

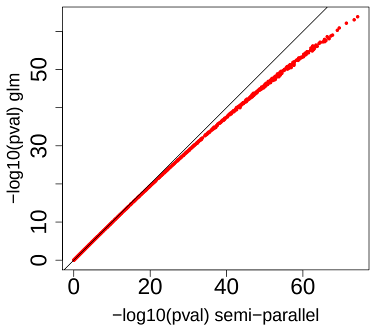
```
p = mean(y)
w = p * (1 - p)
z = log(p / (1 - p)) + (y - p) / w
zc = z - mean(z)
s1 = colSums(S)
s2 = colSums(S ^ 2)
den = s2 - s1 ^ 2/n
b = crossprod(zc, S) / den
err = sqrt(1 / (w[1] * den ))
pval = 2 * pnorm ( -abs(b / err))
```

Speed: 150 Msips. More than 500 times faster than glm

Precision of SP logistic regression

Odds ratios simulated between 1 and 5. Sample size 2000.





Logistic regression with covariates

Similar to linear regression, but incorporating weights.

$$s^* = s - X(X^T W X)^{-1} X^T W s,$$

$$z^* = z - X(X^T W X)^{-1} X^T W z,$$

Solution given by:

$$\beta_1 = \frac{\sum_i w_i z_i^* s_i^*}{\sum_i w_i s_i^{*2}},$$

$$\mathrm{var}(\beta_1) = \frac{1}{\sum_i w_i s_i^{*2}}.$$

Here weights are different between individuals.

SP logistic regression in R

```
mod0 = glm( y ~ X, family = binomial(link  = logit))
p = mod0$fitted
w = p * (1 - p)
z = log(p / (1 - p)) + (y - p)/w
Xtw = t(X * w)
U1 = Xtw %*% z
U2 = solve(Xtw %*% X, U1)
ztr = z - X %*% U2
U3 = Xtw %*% S
U4 = solve(Xtw %*% X, U3)
Str = S - X %*% U4
Str2 = colSums( w * Str ^ 2)
beta1 = crossprod(ztr * w, Str) / Str2
error1 = sqrt(1 / Str2)
pval1 = 2 * pnorm(-abs (beta1/ error1))
```

Logistic regression - speed comparisons

k	glm	SP
1	0.2	57
10	0.1	35
20	0.1	28

10K individuals, 1M SNPs and 10 covariates done within 5 minutes (instead of 28 hours)

Part III: Efficient data access

Efficient data access

- We can do the computations very fast
- But first we need to access the data
- ALL SNP data do not fit into memory
- We need **blocks of SNPs** for all individuals
- Size of the block memory dependent

Reading blocks from text files

- MACH files are just text files
- Typically written as **“row per person”**
- Written on a disc as one chain
- With all SNPs for an individual as a record
- Difficult to access block of SNPs for all individuals
- We could transpose the file and use (slow function) scan
- Does not seem to be efficient solution

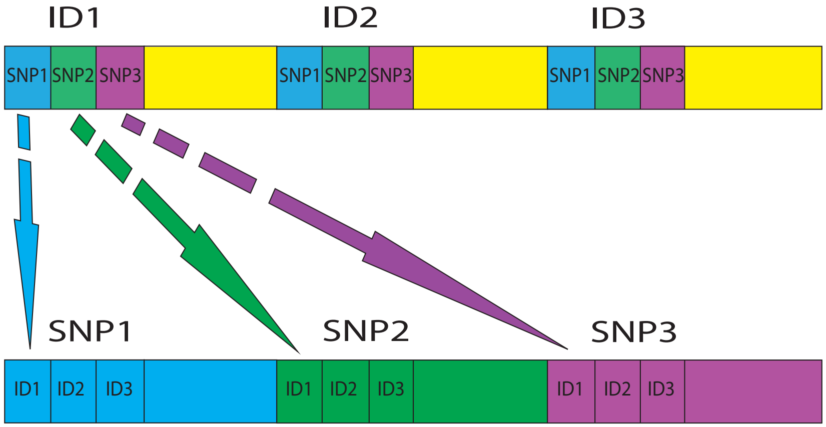
Binary files

- Much faster to access than text files
- Easily created in R (using writeBin and readBin)
- Not convienient because work on vectors
- “row per person” structure is still a problem

Array-oriented binary files

- Saved column-by-column
- A SNP for all individuals makes a record
- In R, **ncdf** and **ff** packages are useful (there are others)

MACH to binary matrix



Package ncdf - writing files

```
library(ncdf)
setwd(" ")
set.seed(2013)
fname = "Ncdf_1.ncdf"
## total number of individuals
N = 10000
### number of individuals that we can read
 from text file at once
n = 1000
### number of SNPs in the file
m = 100000
snps = matrix(2 * runif(m * n), n , m )
```

Ncdf - writing files cont'd

```
# Define dimensions
dimx = dim.def.ncdf("x", "units", 1 : N)
dimy = dim.def.ncdf("y", "units", 1 : m)
# Define variables
varz = var.def.ncdf("z", "numeric", dim = list(dimx, dimy),
                    missval = 999, prec ="short" )
# Create the netCDF file
netf = create.ncdf(fname, vars = list(varz))
#### Store data
snps1 = 100 * snps
nb = N / n
for(i in 1 : nb) {
  k = n  * (i - 1) + 1
  put.var.ncdf(netf, varz, vals = snps1,
               start = c(k, 1), count = c(n, m))
  cat('Block', i, '\n')
}
close(netf)
```

Ncdf - writing cont'd

- It takes 10 minutes to write 100K SNPs for 10K individuals
- We used SSD drive
- 5 times faster than hard disc
- We simulated the data
- Time for "scanning" should be added
- Note that it needs to be done only once
- Take care of details: saving as "short" saves lot of disc space

Ncdf - reading blocks

```
library(ncdf)
setwd(" ")
fname = "Ncdf_1.ncdf"
netf = open.ncdf(fname)
N = 10000
m = 100000
m1 = 10000
nb = m / m1
for(i in 1 : nb){
  t0 = proc.time()[3]
  k = (i - 1) * m1 + 1
  snps_read = get.var.ncdf(netf, "z",
   start = c(1 , k), count = c(N , m1))
  t1 = proc.time()[3] - t0
  cat("Block", i, "read within", t1, "seconds \n")
}
```

Ncdf - reading blocks

- Takes around 4 seconds per block of 10K SNPs
- Which give 400 seconds for 1M SNPs
- Computations take around 200 seconds
- Note that we measured elapsed time here (proc.time()[3])

FF package - writing files

```
library(ff)
setwd(" ")
fname = "ff_1.ff"
N = 10000
n = 1000
m = 100000
snps = matrix(2 * runif(m * n), n , m )

FF = ff(vmode = "short", dim = c(N, m), filename = fname )

snps1 = 100 * snps
nb = N / n
```

FF package - writing files cont'd

```
for(i in 1 : nb){
  k = (i - 1) * n + 1
  l = k + n - 1
  FF[k : l, ] = snps1
  cat("Block" , i, "\n")
}
finalize(FF)
close(FF)
save(FF, file = "ff_1.RData")
```

FF - reading blocks

```
load("ff_1.RData")
nb = m / m1
for(i in 1 : nb) {
  t0 = proc.time()[3]
  k = (i - 1) * m1 + 1
  l = k + m1 - 1
  snps_read = FF[ , k : l]
  t1 = proc.time()[3] - t0
  cat("Block", i, "read within", t1, "seconds \n")
}
```

Both saving and reading twice faster than ncdf

Summary and conclusions

- We made GWAS computations easy
- And feasible on a notebook
- Data access can be a bottleneck
- But array-oriented binary files solve the problem
- More details in our **BMC Bioinformatics** paper
- R codes available on https://bitbucket.org/ksikorska/gwasp

Thank you

&

Good luck with your “GWAS on your notebook”

Please fill out the survey evaluation.
You will be contacted via email.

Today's Presentation Available
http://www.extension.org/pages/68354

Sign up for PBG News
http://pbgworks.org

Sign up for Future Webinars and View Archive
http://www.extension.org/pages/60426