Family-Based Association: PLINK, FaST-LMM, and GCTA

In this exercise you will carry out association analysis of data from a mini genome-wide association study. The data comes from families (related individuals) measured for a quantitative trait of interest. The purpose is to detect which (if any) of the loci are associated with the quantitative trait.

Methodology: we compare standard linear regression (PLINK) against two linear mixed model approaches that account for relatedness between individuals: **FaST-LMM** and **GCTA**. For GCTA we also estimate the heritability accounted for by all genotyped SNPs, and by various subsets of SNPs.

Data overview

Family data consisting of 498 individuals typed at 134,946 SNPs. All individuals have measurements of a quantitative trait of interest; the data has already been QC-ed.

The data is in PLINK binary-file format under `data/`: `quantfamdata.bed` (genotypes), `quantfamdata.bim` (map file), and `quantfamdata.fam` (pedigree structure — the last column gives each individual's quantitative trait value). `data/myphenos.txt` is a ready-made phenotype file derived from `quantfamdata.fam` columns 1, 2, and 6 (family ID, individual ID, trait value) — this is what `--pheno` points to throughout.

How this notebook is organized

Part 1 (Bash kernel) runs every association and heritability analysis end-to-end, producing all the result files used in Part 2:

1. **PLINK** — standard linear regression, ignoring relatedness.
2. **FaST-LMM** — a linear mixed model that accounts for relatedness.
3. **GCTA** `--mlma` — GCTA's own mixed-model association test.
4. **GCTA GREML** — SNP heritability estimation: genome-wide, naively per-chromosome, and then properly partitioned across all 22 autosomes.
5. **GCTA fastGWA** — GCTA's sparse-GRM mixed model, designed for biobank-scale data.

Part 2 (R kernel) reads the result files from sections 1, 2, 3, and 5 above and produces QQ plots, Manhattan plots, and genomic-control (λ) checks for each, plus a direct comparison between GCTA `--mlma` and FaST-LMM, and between GCTA fastGWA and

GCTA `--mlma`. (The GREML heritability estimates from section 4 are read directly off the GCTA log output within Part 1 — there's nothing further to plot for those in Part 2.) You'll switch kernels **once**, right before Part 2 — in JupyterLab's menu bar: **Kernel → Change Kernel… → R**, then continue running cells from there.

Program documentation

- **PLINK**: 1.90 docs
- **R**: r-project.org — from within R, `help()` documents any command
- **FaST-LMM**: fastlmm.github.io while a newer Python `fastlmm` package is available, this notebook instead runs the original `fastlmmc` binary for the exercise.
- **GCTA**: cnsgenomics.com/software/gcta

References

- Chang CC, Chow CC, Tellier LCAM, Vattikuti S, Purcell SM, Lee JJ (2015) Second-generation PLINK: rising to the challenge of larger and richer datasets. GigaScience, 4.
- Lippert C, Listgarten J, Liu Y, Kadie CM, Davidson RI, Heckerman D (2011) FaST linear mixed models for genome-wide association studies. *Nat Methods* 8(10):833-835.
- Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ & Sham PC (2007) PLINK: a toolset for whole-genome association and population-based linkage analysis. *American Journal of Human Genetics*, 81:559-575.
- R Core Team (2025), R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing
- Yang et al. (2011) GCTA: A tool for genome-wide complex trait analysis. *American Journal of Human Genetics*, 88:76-82.

Part 1 — Running the Analyses (Bash kernel)

Everything in this part uses the **Bash** kernel. Each step writes its results to a file, which Part 2 reads back in with R — nothing here depends on R having run first.

> **Kernel:** run the code cells in this section with the **Bash** kernel.

1. PLINK: standard linear regression

To start with, we use PLINK to perform a test equivalent to linear regression analysis, without worrying about the relatedness between individuals. Standard regression assumes each observation is independent — an assumption that's violated here, since this is family data with related individuals who share genetic background. That violation can inflate test statistics and produce false-positive associations, which is exactly what the mixed-model approaches in the rest of this notebook are designed to correct for; PLINK's results here are the "naive" baseline everything else gets compared against.

A copy of the screen output is saved to `plinkresults.log`; the association results themselves go to `plinkresults.qassoc`, one line per SNP, with columns `CHR`, `SNP`, `BP`, `NMISS`, `BETA`, `SE`, `R2`, `T`, and `P` (the most useful being `T`, the test statistic, and `P`, its p-value).

In [ ]:
```
mkdir -p output

ls -l ./data  # sanity check: confirm the input PLINK files are where we exp
# --assoc: simple allelic association test (no relatedness or covariates moc
plink --bfile ./data/quantfamdata --assoc --out output/plinkresults
```

2. FaST-LMM: linear mixed model accounting for relatedness

Now we re-run the analysis using FaST-LMM, which estimates and accounts for relatedness between individuals.

We use the `--bfile quantfamdata` command to tell the program the name (stem) of the files with the input genotype data containing the SNPs to be tested for association, and the `--bfileSim quantfamdata` command to tell the program the name of the files containing the SNPs to be used for estimating relatedness. Here we just use the same files both times, but FaST-LMM would allow us to use different files for these two operations if we prefer.

The command `--pheno` reads the phenotype column from `data/myphenos.txt`. Results are written to `output/FLMMresults`, already ordered by significance, with columns `SNP`, `Chromosome`, `Position`, `Pvalue` (among others).

In [ ]:
```
# Runs the actual fastlmmc binary; -bfileSim gives the genotype data used to
fastlmmc -bfile ./data/quantfamdata -pheno ./data/myphenos.txt -bfileSim ./d
```

3. GCTA `--mlma`: GCTA's own mixed-model association test

GCTA is designed primarily for apparently-unrelated individuals, but we apply it here to our related individuals for comparison with FaST-LMM. `--mlma` performs mixed-linear-

model association analysis, fitting the same kind of genetic random effect described above; `--bfile` and `--pheno` give the genotype and phenotype data (`data/myphenos.txt`, derived from `quantfamdata.fam` as described above).

In [ ]:
```
# --mlma: mixed-linear-model association test (GCTA estimates relatedness in
gcta64 --mlma --bfile ./data/quantfamdata --pheno ./data/myphenos.txt --out
```

4. GCTA: SNP heritability (GREML)

GCTA can be used to estimate the heritability accounted for by all autosomal genome-wide SNPs.

To estimate the heritability, we first build a genetic relationship matrix (GRM) — from all the autosomal significant SNPs. We then the GREML methof (genomic-relatedness-based REML) to fit a mixed model with the GRM as a random effect and estimate the (SNP) heritability.

In [ ]:
```
# 1. Build the genetic relationship matrix (GRM) from all autosomal SNPs
gcta64 --bfile ./data/quantfamdata --autosome --make-grm-bin --out output/GC
# 2. Estimate heritability (REML) using that GRM
gcta64 --reml --grm-bin output/GCTAgrm --pheno ./data/myphenos.txt --out out
```

Per-chromosome heritability (naive)

Repeating this separately for chromosomes 1, 2, 6, and 12: expect chromosomes 1 and 2 to look non-significant given their standard errors, while 6 and 12 (where the earlier association signals were) look significant. Note the four chromosome-specific estimates sum to *more* than the overall heritability — with related individuals, single-chromosome GRMs can be confounded by shared non-genetic effects or correlations with other chromosomes, inflating each chromosome's apparent contribution.

In [ ]:
```
# Chromosome 1
gcta64 --bfile ./data/quantfamdata --chr 1 --make-grm-bin --out output/GCTAg
gcta64 --reml --grm-bin output/GCTAgrmchr1 --pheno ./data/myphenos.txt \
--out output/GCTAheritchr1

# Chromosome 2
gcta64 --bfile ./data/quantfamdata --chr 2 --make-grm-bin --out output/GCTAg
gcta64 --reml --grm-bin output/GCTAgrmchr2 --pheno ./data/myphenos.txt \
--out output/GCTAheritchr2

# Chromosome 6
gcta64 --bfile ./data/quantfamdata --chr 6 --make-grm-bin --out output/GCTAg
gcta64 --reml --grm-bin output/GCTAgrmchr6 --pheno ./data/myphenos.txt --out

# Chromosome 12
gcta64 --bfile ./data/quantfamdata --chr 12 --make-grm-bin --out output/GCTA
gcta64 --reml --grm-bin output/GCTAgrmchr12 --pheno ./data/myphenos.txt --ou
```

Correctly partitioning heritability across all 22 autosomes

To partition heritability properly we need chromosome-specific GRMs for *every* autosome, included simultaneously in one model — not estimated separately as above. First build the remaining per-chromosome GRMs:

In [ ]:
```
# Same two-step (build-GRM) pattern as above, repeated for each remaining au
# (chromosomes 1, 2, 6, and 12 were already built above)
gcta64 --bfile ./data/quantfamdata --chr 3 --make-grm-bin --out output/GCTAg
gcta64 --bfile ./data/quantfamdata --chr 4 --make-grm-bin --out output/GCTAg
gcta64 --bfile ./data/quantfamdata --chr 5 --make-grm-bin --out output/GCTAg
gcta64 --bfile ./data/quantfamdata --chr 7 --make-grm-bin --out output/GCTAg
gcta64 --bfile ./data/quantfamdata --chr 8 --make-grm-bin --out output/GCTAg
gcta64 --bfile ./data/quantfamdata --chr 9 --make-grm-bin --out output/GCTAg
gcta64 --bfile ./data/quantfamdata --chr 10 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 11 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 13 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 14 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 15 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 16 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 17 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 18 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 19 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 20 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 21 --make-grm-bin --out output/GCTA
gcta64 --bfile ./data/quantfamdata --chr 22 --make-grm-bin --out output/GCTA
```

Then run REML with all 22 chromosome-specific GRMs simultaneously (`data/multipleGRMs.txt` lists their stem names — take a look at it). In this dataset the analysis fails to converge, likely because this kind of full-partition analysis really needs a larger set of less closely related individuals than we have here.

As a more tractable alternative, we partition heritability into just two sets — chromosome 6 versus everything else. First combine the non-chr6 GRMs into one (`data/multipleGRMsnot6.txt` lists their stems), then run REML with both pieces (`data/multipleGRMs6andnot6.txt` lists chr6 and the combined rest). Expect a total heritability of about 0.47, split roughly 0.29 for chromosome 6 and 0.17 for the remaining autosomes.

In [ ]:
```
# Attempt REML with all 22 per-chromosome GRMs at once (expected not to conv
gcta64 --reml --mgrm-bin ./data/multipleGRMs.txt --pheno ./data/myphenos.txt

# Combine the 21 non-chr6 GRMs into a single merged GRM
gcta64 --mgrm-bin ./data/multipleGRMsnot6.txt --make-grm --out output/GCTAgr

# REML with just two variance components: chr6 alone, and the combined rest
gcta64 --reml --mgrm-bin ./data/multipleGRMs6andnot6.txt --pheno ./data/myph
--out output/GCTAherit6andnot6
```

5. GCTA fastGWA: sparse-GRM mixed model for biobank-scale data

fastGWA is GCTA's alternative MLM-based GWAS method, designed for biobank-scale datasets (e.g. UK Biobank) — we apply it here to our much smaller dataset for comparison. The dense GRM used by `--mlma` and GREML above stores every pairwise relationship, so it scales quadratically with sample size — impractical once you're into the hundreds of thousands of individuals. fastGWA's sparse-GRM approximation is what makes mixed-model association testing feasible at that scale; `--fastGWA-mlm` (the command used below) is the actual association test, paired with the sparse GRM built in this section.

Sparse GRM

A sparse GRM keeps only relationship pairs above a cutoff (here 0.05, zeroing out everything below it), which is far cheaper to work with at scale than the full dense GRM — in a large cohort, most pairs of individuals are unrelated (relatedness ≈ 0) anyway, so storing only the meaningfully nonzero entries avoids computing and storing an N×N matrix that's mostly noise.

In [ ]:
```
# Keep only relationship pairs > 0.05; everything below that is treated as u
gcta64 --grm output/GCTAgrm --make-bK-sparse 0.05 --out output/GCTAsparsegrm
```

Partitioned GRM computation

For application using real biobank-scale data, computing the full dense GRM in one pass can be computationally prohibitive. GCTA can partition the computation into *m* parts (by row) and compute each part separately with `--make-grm-part m i`, to be merged back together afterward. Here we split into 3 parts (using 5 threads each) purely to demonstrate the workflow, then merge the parts and build a second sparse GRM from the result — its top lines should match `GCTAsparsegrm.grm.sp` from above.

In [ ]:
```
# Compute the GRM in 3 separate row-partitions (parts 1-3 of 3, 5 threads ea
# all three must finish before they're merged in the next cell
gcta64 --bfile ./data/quantfamdata --make-grm-part 3 1 --thread-num 5 \
--out output/test
gcta64 --bfile ./data/quantfamdata --make-grm-part 3 2 --thread-num 5 \
--out output/test
gcta64 --bfile ./data/quantfamdata --make-grm-part 3 3 --thread-num 5 \
--out output/test
```

In [ ]:
```
# Each partition writes its own output/test.part_3_<i> files; GCTA's GRM for
# just stacked per-pair records, so concatenating the parts reassembles the
cat output/test.part_3_*.grm.id > output/test.grm.id
```

```
cat output/test.part_3_*.grm.bin > output/test.grm.bin
cat output/test.part_3_*.grm.N.bin > output/test.grm.N.bin
```

In [ ]:
```
# Build a sparse GRM from the merged (partitioned) GRM — should match GCTAsp
gcta64 --grm output/test --make-bK-sparse 0.05 --out output/newsparsegrm

head output/*.sp  # preview both sparse GRMs' top lines to compare
```

Principal components and fastGWA-mlm

For the association analysis, we'd normally use the sparse GRM to model close relationships as random effects, with principal components included as fixed effects. First compute the top 5 PCs:

In [ ]:
```
# Compute the top 5 principal components from the (dense) GRM
gcta64 --grm-bin output/GCTAgrm --pca 5 --out output/pcs
```

Now run fastGWA-mlm four ways: with each of the two sparse GRMs (`GCTAsparsegrm` and `newsparsegrm`), both with and without the PCs as covariates. Including the PCs (`--qcovar pcs.eigenvec`) tends to make the genetic variance estimate Vg non-significant, causing fastGWA to silently fall back to plain linear regression — not what we want here — so the no-covariate runs are the ones that actually force a linear mixed model.

In [ ]:
```
# 1. GCTAsparsegrm + PC covariates
gcta64 --bfile ./data/quantfamdata --grm-sparse output/GCTAsparsegrm \
--fastGWA-mlm --pheno ./data/myphenos.txt --qcovar output/pcs.eigenvec \
--out output/sparse_assoc

# 2. newsparsegrm (partitioned/merged version) + PC covariates
gcta64 --bfile ./data/quantfamdata --grm-sparse output/newsparsegrm \
--fastGWA-mlm --pheno ./data/myphenos.txt --qcovar output/pcs.eigenvec \
--out output/newsparse_assoc

# 3. GCTAsparsegrm, no covariates — forces an actual mixed model (see note a
gcta64 --bfile ./data/quantfamdata --grm-sparse output/GCTAsparsegrm \
--fastGWA-mlm --pheno ./data/myphenos.txt --out output/LMMsparse_assoc

# 4. newsparsegrm, no covariates
gcta64 --bfile ./data/quantfamdata --grm-sparse output/newsparsegrm \
--fastGWA-mlm --pheno ./data/myphenos.txt --out output/LMMnewsparse_assoc
```

Part 2 — Reading Results and Plotting

Switch kernels now: in JupyterLab's menu bar, go to **Kernel → Change Kernel… → R**, then continue running cells from here. Everything below reads files already written by

Part 1, so nothing needs to be re-run in Bash.

> **Kernel:** run the code cells in this section with the **R** kernel.

In [ ]:
```r
# Load the manhattan()/qq() plotting helper functions used throughout Part 2
source("scripts/qqmanHJCupdated.R")
```

PLINK results

134,946 rows, one per SNP — too many to eyeball, so we visualize with a Manhattan plot (colored by chromosome) and a QQ plot. Visually, chromosomes 6 and 12 stand out, with possibly chromosome 5 too.

The genomic control inflation factor λ converts p-values to 1-df chi-squared statistics and takes their median, scaled by 0.456 (Devlin & Roeder, 1999) — values near 1.0 indicate the bulk of results are consistent with the null. Expect **λ ≈ 1.10** here (mild inflation from relatedness, since plain PLINK doesn't account for it).

In [ ]:
```r
# Read PLINK's association results and preview the columns
res1 <- read.table("output/plinkresults.qassoc", header = T)
head(res1)
```

In [ ]:
```r
# Manhattan plot: -log10(P) by genomic position, colored by chromosome (opti
png("output/mh1.png")
manhattan(res1,
    pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5
)
dev.off()

# QQ plot: observed vs. expected -log10(P) under the null (should hug the di
png("output/qq1.png")
qq(res1$P)
dev.off()
```

In [ ]:
```r
# Genomic control inflation factor: convert p-values to 1-df chi-squared sta
# take the median, and scale by the expected null median (0.456) — see note
chi <- (qchisq(1 - res1$P, 1))
lambda <- median(chi) / 0.456
lambda
```

FaST-LMM results

The p-value column here is named `Pvalue`, and results are already ordered by significance. Expect **λ ≈ 0.99** — noticeably less inflated than PLINK's regression, since FaST-LMM accounts for relatedness. (This run doesn't use leave-one-chromosome-out, matching the original exercise's own `fastlmmc` invocation — see the note in Part 1.)

In [ ]:
```
# Read FaST-LMM's association results (tab-delimited) and preview the columns
res2 <- read.table("output/FLMMresults", header = T, sep = "\t")
head(res2)
```

In [ ]:
```
# Genomic control lambda for FaST-LMM (see the PLINK results section above f
chi <- (qchisq(1 - res2$Pvalue, 1))
lambda <- median(chi) / 0.456
lambda
```

In [ ]:
```
# manhattan()/qq() expect columns named SNP, CHR, BP, P — rename FaST-LMM's
new <- data.frame(res2$SNP, res2$Chromosome, res2$Position, res2$Pvalue)
names(new) <- c("SNP", "CHR", "BP", "P")
head(new)

png("output/qq2.png")
qq(new$P)
dev.off()

png("output/mh2.png")
manhattan(new,
    pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5
)
dev.off()
```

GCTA `--mlma` results

Expect λ close to 1.0, and QQ/Manhattan plots similar to FaST-LMM's.

In [ ]:
```
# Read GCTA --mlma's association results and preview the columns
res3 <- read.table("output/GCTAresults.mlma", header = T)
head(res3)
```

In [ ]:
```
# Genomic control lambda for GCTA --mlma
chi <- (qchisq(1 - res3$p, 1))
lambda <- median(chi) / 0.456
lambda
```

In [ ]:
```
# Rename to the SNP/CHR/BP/P columns manhattan()/qq() expect (see the FaST-L
new3 <- data.frame(res3$SNP, res3$Chr, res3$bp, res3$p)
names(new3) <- c("SNP", "CHR", "BP", "P")
head(new3)

png("output/qq3.png")
qq(new3$P)
dev.off()

png("output/mh3.png")
manhattan(new3,
    pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5
```

```
)
dev.off()
```

Comparing GCTA to FaST-LMM

Expect the two methods' -log10(P) values to line up closely along the diagonal — both are mixed models over the same data without a LOCO guard, so any gap here reflects implementation differences (e.g. GCTA's exact vs. FaST-LMM's spectral variance-component estimation) rather than a methodological one.

In [ ]:
```
# Join GCTA's and FaST-LMM's results on SNP so we can compare p-values for t
new2 <- data.frame(res2$SNP, res2$Chromosome, res2$Position, res2$Pvalue)
names(new2) <- c("SNP", "CHR", "BP", "P")
merged <- merge(new3, new2, by = "SNP", sort = F)

head(res2)
head(new2)
head(new3)
head(merged)
```

In [ ]:
```
# Scatter of -log10(P): the diagonal line marks perfect agreement (y = x), r
png("output/compareGCTAFLMM.png")
plot(-log10(merged$P.x), -log10(merged$P.y))
abline(0, 1)
dev.off()
```

GCTA fastGWA results

Four result sets from Part 1: sparse GRM with/without PC covariates, for each of the two (equivalent) sparse GRMs we built. The linear-mixed-model runs (no PC covariates, forced to actually use the sparse GRM) should show λ noticeably closer to 1.0 than the PC-covariate runs, which fastGWA silently downgraded to plain linear regression.

In [ ]:
```
# res4/res5: sparse GRM + PC covariates (fastGWA silently falls back to line
# res6/res7: same two GRMs, no covariates (forces an actual mixed model)
res4 <- read.table("output/sparse_assoc.fastGWA", header = T)
res5 <- read.table("output/newsparse_assoc.fastGWA", header = T)
res6 <- read.table("output/LMMsparse_assoc.fastGWA", header = T)
res7 <- read.table("output/LMMnewsparse_assoc.fastGWA", header = T)

head(res4)
head(res5)
head(res6)
head(res7)
```

In [ ]:
```
# res4: sparse GRM + PC covariates
chi <- (qchisq(1 - res4$P, 1))
lambda <- median(chi) / 0.456
lambda
# res5: newsparsegrm + PC covariates
```

```
chi <- (qchisq(1 - res5$P, 1))
lambda <- median(chi) / 0.456
lambda
# res6: sparse GRM, no covariates (forced mixed model)
chi <- (qchisq(1 - res6$P, 1))
lambda <- median(chi) / 0.456
lambda
# res7: newsparsegrm, no covariates (forced mixed model)
chi <- (qchisq(1 - res7$P, 1))
lambda <- median(chi) / 0.456
lambda
```

In [ ]:
```
# res4: sparse GRM + PC covariates
new4 <- data.frame(res4$SNP, res4$CHR, res4$POS, res4$P)
names(new4) <- c("SNP", "CHR", "BP", "P")

png("output/qq4.png")
qq(new4$P)
dev.off()

png("output/mh4.png")
manhattan(new4, pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5)
dev.off()
```

In [ ]:
```
# res5: newsparsegrm (partitioned/merged version) + PC covariates
new5 <- data.frame(res5$SNP, res5$CHR, res5$POS, res5$P)
names(new5) <- c("SNP", "CHR", "BP", "P")

png("output/qq5.png")
qq(new5$P)
dev.off()

png("output/mh5.png")
manhattan(new5, pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5)
dev.off()
```

In [ ]:
```
# res6: sparse GRM, no covariates (forced mixed model)
new6 <- data.frame(res6$SNP, res6$CHR, res6$POS, res6$P)
names(new6) <- c("SNP", "CHR", "BP", "P")

png("output/qq6.png")
qq(new6$P)
dev.off()

png("output/mh6.png")
manhattan(new6, pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5)
dev.off()
```

In [ ]:
```
# res7: newsparsegrm, no covariates (forced mixed model)
new7 <- data.frame(res7$SNP, res7$CHR, res7$POS, res7$P)
names(new7) <- c("SNP", "CHR", "BP", "P")
```

```
png("output/qq7.png")
qq(new7$P)
dev.off()

png("output/mh7.png")
manhattan(new7, pch = 20, suggestiveline = F, genomewideline = F, ymin = 2,
    cex.x.axis = 0.65, colors = c("black", "dodgerblue"), cex = 0.5)
dev.off()
```

Comparing GCTA fastGWA to GCTA `--mlma`

Finally, compare the fastGWA linear-mixed-model result (sparse GRM, recalculated version, no PC covariates) against the original `--mlma` result. Expect these to line up extremely closely.

In [ ]:
```
# As above: points along the y = x line indicate the two methods agree close
png("output/compareGCTAfastGWA.png")
plot(-log10(res3$p), -log10(res7$P))
abline(0, 1)
dev.off()
```

Other packages

- Comparable to FaST-LMM: EMMAX, GEMMA, MMM, GenABEL, Mendel
- Comparable to GCTA: DISSECT