Manuscript Title

This manuscript (permalink) was automatically generated from related-sciences/gwas-analysis-manuscript@2dacdac on February 17, 2020.

Authors

- **John Doe**
  XXXX-XXXX-XXXX-XXXX · johndoe · johndoe
  Department of Something, University of Whatever · Funded by Grant XXXXXXXX

- **Jane Roe**
  XXXX-XXXX-XXXX-XXXX · janeroe
  Department of Something, University of Whatever; Department of Whatever, University of Something

Abstract

Outline

- Background
  - Scale of current genomic datasets
    - Stats on number of Biobanks
    - Stats on UKBB
    - Stats on growth from HapMap and 1KG for comparison
  - Focus of paper
    - Genotyping array data QC, control flow, and association modeling
    - Scalability problems with single-node, in-memory tools
    - Spark
  - Tutorial
    - Explain necessary GWAS toolkit operations
      - Data management (filtering, merging, etc.)
      - Liftover
      - Summary statistics (HWE, AF, call rates, heterozygosity, etc.)
      - Population stratification (IBS + MDS)
      - Association analysis (logreg, fisher/chisq)
      - Genetic relatedness
    - Introduce Marees 2018 paper and explain how tutorial is re-implemented with other tools
  - Tools
    - Primary: PLINK, Glow, Hail
    - Secondary: dask, bigsnpr, scikit-allel, pysnptools/fastlmm
      - Modin may be worth mentioning, even though out-of-core isn't really supported
  - Datasets
    - HapMap
    - 1KG
    - 3K rice genome
  - Data Formats
    - plink, bgen, parquet, Hail MT, vcf, hdf5/npz/zarr
    - Explain encoding and compression concerns
- Results
  - Code
    - Figure: flow chart of Marees analysis
      - Explain what the resulting code for this project does
    - Figure: side-by-side comparison of code examples
      - Primary differences:
        - Hail is an API over opaque data structures and implementations
        - Glow is simply a convention for representing genetic data in a Spark Dataset, with accompanying methods
        - PLINK is a gigantic list of parameterized commands for a single-core, single-node CLI
      - Operation differences
        - Operations that are similarly easy with all 3 toolkits:
          - call rate filtering
          - heterozygosity rate filtering
          - HWE filtering
          - AF filtering
        - LD Pruning: non-existent in Glow, very slow in Hail
          - However, Glow does support inline calls to PLINK (albeit very awkwardly since PLINK is not streaming software)
        - Gender imputation: PLINK=automatic, Hail=automatic, Glow=manual
          - PLINK and Hail both use inbreeding coefficient on X chromosome data
          - Glow approach (and Marees 2018) look at homozygosity rate on X instead
        - Liftover
          - Not available in PLINK
          - Hail supports coordinate liftover only (not variant liftover)
            - Requires a chain file for the destination reference genome
          - Glow supports both coordinate liftover and variant liftover
            - Requires a chain file and a reference fasta for the destination genome

            - see the notebook at https://glow.readthedocs.io/en/latest/etl/lift-over.html for specification of chain and reference files
        - PCA for population stratification: simple in PLINK and Hail, non-existent in Glow
    - Usability:
        - Extending and learning algorithms from Glow source code is the easiest of all tools
        - Hail documentation is fairly thorough, though essentially no examples or answers are available outside of that documentation (or the discourse)
        - PLINK examples and documentation are both very extensive and easily found, however few PLINK workflows don't also include the need for some scripting language in the same pipeline for interpreting/visualizing results (those outputs are then often used to parameterize other PLINK commands, as exemplified by the tutorials in the project)
- Data
    - Figure: File Format Comparison
        - Show comparison of file sizes by dataset and format
- Performance
    - Figure: Times for operations by dataset and toolkit
    - Touch on vectorization support in breeze (some ops use jni to LAPACK, but simpler ones like sums do not) as compared to numpy
    - Explain benefits of bitpacking (modify 1KG dask nb to have step without GeneticBitPacking filter and compare to original)
- Discussion
    - Computational operations needed in GWAS analyses (see "Computational Operations" in notes)
        - This may be a good place to characterize all operations and what matrix functions support them

Notes

UKBB

- UKB Genotyping and Imputation Data Release March 2018 – FAQ
    - Imputed call set is 2.1T bgen
    - Measured call set is 92G bed
- DETAILS AND CONSIDERATIONS OF THE UK BIOBANK GWAS
    - "Starting from the 487,409 individuals with phased and imputed genotype data, we filtered down to 337,199 QC positive individuals"
    - "We have run 1513 unique phenotypes"
    - "Over 92 million imputed autosomal SNPs were available for analysis. As of this writing, over half of these SNPs were undergoing re-imputation by the UK Biobank team due to mapping issues, leaving the ~40 autosomal million SNPs imputed from the Haplotype Reference Consortium as eligible for analysis. We further restricted to SNPs with minor allele frequency (MAF) > 0.1% and HWE p-value > 1e-10 in the 337,199 QC positive individuals, an INFO score > 0.8 (directly from UK Biobank), leaving 10.8 million SNPs for analysis."
- http://biobank.ctsu.ox.ac.uk/crystal/label.cgi?id=100314
    - Genotypes and imputation therefrom for 488,000 participants
    - Exome sequences for 50,000 participants
    - Whole genome sequences for 50 participants
- Quality control guide: http://www.ukbiobank.ac.uk/wp-content/uploads/2014/04/UKBiobank_genotyping_QC_documentation-web-1.pdf
    - UK Biobank team uses PLINK (for 1KG mostly), shellfish, and flashPCA for QC that is then used by NealeLab
    - This guide is about QC for interm release of 150k samples
        - Results in 152k samples by 806k variants
    - SNP QC is first performed only on European cohort
        - Using a homogeneous cohort is ideal just for initial SNP filters
    - SNPs are filtered based on heterozygosity (adjusted by regression using cohort since heterozygosity differs so much), missingness, and HWE
    - QC is done for each SNP in 4.8k sample batches (matching Affymetrix batches) so a SNP can fail QC in one batch and become all missing
    - SNP QC also done using ROH which is sequential heterozygous call rates
    - Sample QC adjusts for gender (finds .1% are wrong – has interesting aneuploidy figure)
- GPU Acceleration
- Epistatic Interactions
    - SHEsisEpi, a GPU-enhanced genome-wide SNP-SNP interaction scanning algorithm, efficiently reveals the risk genetic epistasis in bipolar disorder

    - 1000× faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis
        - Uses reparameterization of Newton's method in PLINK to make interaction model much faster to solve
- IBS/IBD
    - PlinkGPU: A Framework for GPU Acceleration of Whole Genome Data Analysis
        - Ports pairwise IBS distance calculation in PLINK to GPU using block-wise calculations
        - Summary of PLINK capabilities:
            - Data management (filtering, merging, etc.)
            - Summary statistics (HWE, AF, call rates, heterozygosity, etc.)
            - Population stratification (IBS + MDS)
            - Association analysis (logreg, fisher/chisq)
            - Genetic relatedness

Ecosystem Tools

- bigsnpr
    - https://privefl.github.io/bigsnpr/articles/demo.html
    - Supports PLINK and UKBB BGEN reading
    - Efficient analysis of large-scale genome-wide data with two R packages: bigstatsr and bigsnpr
    - bigstatsr is related but more generic library for out-of-core matrix ops, PCA, and linear models
    - Supports imputation using custom XGBoost model as well as wrappers for calls out to PLINK + Beagle
- FaST-LMM
    - From 2011 Nature Methods paper FaST linear mixed models for genome-wide association studies
    - PySnpTools
        - https://fastlmm.github.io/PySnpTools/#snpreader-snpreader
        - Can read and write PLINK bed/ped data (it appears to be designed specifically for working efficiently with PLINK data)
        - Supports "DistributedBed" files with chunked PLINK datasets
        - Supports slicing of PLINK datasets (i.e. random IO)
        - Supports IO with npz, hdf5, and memmap files
        - Has efficient c++ reader/writer implementations
- scikit-allel
    - In memory EDA for genotyping data
    - Supports vcf, plink,
    - Has vcf_to_{npz,zarr,recarray} methods
    - Represents genotype calls as 3D uint8 arrays
        - GenotypeArray > This class represents data on discrete genotype calls as a 3-dimensional numpy array of integers. By convention the first dimension corresponds to the variants genotyped, the second dimension corresponds to the samples genotyped, and the third dimension corresponds to the ploidy of the samples.

          Each integer within the array corresponds to an allele index, where 0 is the reference allele, 1 is the first alternate allele, 2 is the second alternate allele, … and -1 (or any other negative integer) is a missing allele call. A single byte integer dtype (int8) can represent up to 127 distinct alleles, which is usually sufficient. The actual alleles (i.e., the alternate nucleotide sequences) and the physical positions of the variants within the genome of an organism are stored in separate arrays, discussed elsewhere.

          In many cases the number of distinct alleles for each variant is small, e.g., less than 10, or even 2 (all variants are biallelic). In these cases a genotype array is not the most compact way of storing genotype data in memory. This class defines functions for bit-packing diploid genotype calls into single bytes, and for transforming genotype arrays into sparse matrices, which can assist in cases where memory usage needs to be minimised. Note however that these more compact representations do not allow the same flexibility in terms of using numpy universal functions to access and manipulate data.
    - Supports bitpacking by collapsing the ploidy dimension (axis=2) into a single byte using 4 bits for each uint8

- Phasing is supported by assuming the ordering in the ploidy dimension has meaning: > If the genotype calls are unphased then the ordering of alleles along the third (ploidy) dimension is arbitrary

- http://alimanfoo.github.io/2016/06/10/scikit-allel-tour.html > The scikit-allel genotype array convention is flexible, allowing for multiallelic and polyploid genotype calls. However, it is not very compact, requiring 2 bytes of memory for each call. A set of calls for 10,000,000 SNPs in 1,000 samples thus requires 20G of memory.

  One option to work with large arrays is to use bit-packing, i.e., to pack two or more items of data into a single byte. E.g., this is what the plink BED format does. If you have have diploid calls that are only ever biallelic, then it is possible to fit 4 genotype calls into a single byte. This is 8 times smaller than the NumPy unpacked representation.

  However, coding against bit-packed data is not very convenient. Also, there are several libraries available for Python which allow N-dimensional arrays to be stored using compression: h5py, bcolz and zarr. Genotype data is usually extremely compressible due to sparsity - most calls are homozygous ref, i.e., (0, 0), so there are a lot of zeros.

LD pruning

- https://hail.is/docs/0.1/hail.VariantDataset.html#hail.VariantDataset.ld_prune
- Hail discussions on ld prune implementation:
  - https://discuss.hail.is/t/ld-prune-implementation-in-0-1/244
  - https://dev.hail.is/t/seeking-some-input-on-current-implementation-of-ld-prune-method/78/11
- Hail 0.2 implementation (https://hail.is/docs/0.2/_modules/hail/methods/statgen.html#ld_prune)
  - Local LD prune implementation: https://github.com/hail-is/hail/blob/master/hail/src/main/scala/is/hail/methods/LocalLDPrune.scala#L227
  - Summary of Hail 0.2 implementation:
    - A greedy strategy first prunes using double loop for all variants in a single partition and considers only R2 (no MAF) so the choice is arbitrary
      - This is what is ultimately returned, though everything that follows is intended to identify correlated variants spanning partitions that need to be removed
    - Next, a giant BlockMatrix multiplication with its transpose used to get R2 across all partitions
    - This matrix is than sparsified based on row intervals (see sparsify_row_intervals)
    - The matrix is collapsed to entries in processed further as (variant i, variant j, r2)
    - Variants *above* the r2 threshold are kept are kept and fed into a maximally independent set finder
      - MIS is done in memory on a single machine (see Graph.scala)
    - The final set of variants to remove is filtered out of the result from step 1
  - Operations needed to recreate this:
    - Matrix transpose + multiplication
      - This is how the LD calculation works
    - Matrix + vector subtraction with broadcasting and element-wise matrix multiplication
      - The sparsification of the LD matrix is done by determining which variants are within some number of bp from each other
      - This is done in a custom Scala routine, but it could also be expressed as a subtraction of genomic positions to create a mask matrix
      - The mask matrix could then be multiplied by the LD matrix to get the correct banding
    - Matrix enumeration (rows, cols, or entries)
      - In the Hail case, enumeration is used to collapse a sparse LD matrix, filter it per entry, build a list of graph edges, and then run a custom scala routine to find maximal independent sets
      - If Privé's clumping was used instead, there is a way to express everything with per calculations on the sparsified LD matrix along with distributed sorting and filtering (having an upfront "importance" metric to sort by makes it much easier to do without any single-node algorithm bottlenecks)
- https://www.cog-genomics.org/plink/1.9/ld
- Actual code from C Chang on how ld pruning works:
  - https://groups.google.com/d/msg/plink2-users/w5TuZo2fgsQ/WbNnE16_xDIJ
- In PLINK:
  - (PLINK 1.07) https://wlz0726.github.io/2017/05/27/LD-prune-with-plink/
    - a. consider a window of 50 SNPs
    - b. calculate LD between each pair of SNPs in the window
    - b. remove one of a pair of SNPs if the LD is greater than 0.5
    - c. shift the window 5 SNPs forward and repeat the procedure

PCA

- Projecting samples using pre-computed PCs
  - Hail does not have this but McArthur lab has examples (see discourse post and linked code)
- On how loadings can be used to identify poor PCA results
  - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2912642/
  - "Correlated SNPs will therefore have high loadings, because correlated random variables can generate linear combinations with high variability. As we demonstrate, the net effect is to give higher weight to groups of correlated SNPs, although there is little reason to believe that such SNPs will perform well in differentiating among subpopulations. An intermediate goal, therefore, is to eliminate the distorting effect of the redundant information provided by groups of highly correlated SNP genotypes."
- HWE normalization
  - https://doc.goldenhelix.com/SVS/latest/svsmanual/ftParts/pca.html#formulas-for-pca-normalization-of-genotypic-data
  - Explained in Population Structure and Eigenanalysis
- Identifying outlier SNPs in loadings
  - Peaks in loadings may indicate LD structure captured by PCA
  - A Practical Approach to Adjusting for Population Stratification in Genome-wide Association Studies: Principal Components And Propensity Scores (PCAPS)
    - "Most importantly, patterns of local linkage disequilibrium (LD) may cause PCA to create "nuisance axes", which may be interpreted as the existence of subpopulations that reflect localized LD phenomena rather than plausible PS (Zou et al. 2010)"
  - They may also indicate natural selection underway within multiple populations
    - pcadapt: an R package to perform genome scans for selection based on principal component analysis
    - Regresses PCs against each SNP to see which SNPs aren't explained by PCA
    - "The first statistic based on PCA was the communality statistic, which measures the percentage of variation of a single nucleotide polymorphism (SNP) explained by the first K principal components"
  - One interpretation appears to be that SNPs not captured by PCs may represent selection
    - Detecting Genomic Signatures of Natural Selection with Principal Component Analysis: Application to the 1000 Genomes Data
- From Population Structure and Eigenanalysis: - Defines a test for stating whether population structure exists in a bi-allelic dataset - Also gives a test to answer the question "Does the data show evidence of additional population structure over and above what has already been detected?"
- Types of population structure (Li et al. 2010)
  - Discrete
  - Admixed
  - Hierarchical
- PCA Loadings
  - Genome-wide SNP and haplotype analyses reveal a rich history underlying dog domestication (vonHoldt 2010)
    - Uses small numbers of highly ranked SNPs (by loading) to uniquely define genetic differences between dogs and grey wolves (supplemental table 2)
- Phylogenetic Analysis
  - See Jombart et al. 2010 (ppca which uses PCA and clustering to identify ancestry

Ascertainment bias

- SNP arrays are designed using an "ascertainment sample" and it is this sample that determines what SNPs make it onto the array
- As these SNPs are often filtered to only this with AF in a certain range, it is possible for many rarer variants in such a small sample to be ignored (ascertainment bias)
- There are ways to try to adjust for this, but it is not possible when the "ascertainment sample" does not match the ancestry of the "typed sample" (the one an experiment is being run on)
  - From Population genetic analysis of ascertained SNP data (Nielson 2004):
    - "In cases where there is little or no overlap between the ethnicities of the individuals included in the typed sample and the ascertainment samples, however, corrections can only be made in parametric models describing the genetic relationship between the populations. In such cases, it will typically be difficult or impossible to use classical non-parametric methods for statistical inference."

File Formats

- BGEN
  - BGEN: a binary file format for imputed genotype and haplotype data

    - https://www.cog-genomics.org/plink/2.0/formats#bgen
    - https://www.well.ox.ac.uk/~gav/bgen_format/
    - https://www.well.ox.ac.uk/~gav/bgen_format/spec/latest.html
- SIMD and Native Libraries
- Hail uses breeze (https://github.com/hail-is/hail/blob/de2968e0bf9215e058b1fbace4ae618cc93463fe/hail/src/main/scala/is/hail/expr/ir/BlockMatrixIR.scala)
- Breeze uses netlib-java which is a wrapper for BLAS/LAPACK/ARPACK (https://github.com/scalanlp/breeze/wiki/Breeze-Linear-Algebra#performance)
    - Breeze uses LAPACK for some ops like svd but not others like sum
    - Sums along an exist apparently don't exist in LAPACK (see this post)
    - Matrix multiplication (DenseMatrixOps) seems to use BLAS though
- Hail says atlas should be installed for native linear algebra: https://hail.is/docs/0.2/getting_started.html#blas-and-lapack
- SIMD support doesn't really seem to exist in spark 2.x, or at least this jira ticket states that companies are hacking this functionality in now (3.x might have it)
- A decent explanation of the relationship between breeze and LAPACK/BLAS: http://izmailoff.github.io/ml/neural-network-from-scratch-in-scala/
- On Atlas vs BLAS vs LAPACK: "ATLAS is a portable reasonably good implementation of the BLAS interfaces, that also implements a few of the most commonly used LAPACK operations."
- An OpenJDK vector API seems like it would provide SIMD op access, though it is still incubating as of JDK13 (example)
- Good overview of vectorization in Spark: https://www.waitingforcode.com/apache-spark-sql/vectorized-operations-apache-spark-sql/read

C/G and A/T SNPs

- This QC step is common because the probe sequence on genotyping arrays is not specific to a reference genome, by design. This means that the genotyping data can tell you that an "A" nucleotide was present at a locus but it doesn't actually know if this nucleotide represents some kind of "variant" with respect to a larger population. The probes are chosen such that they capture individual sites of common variation but deciding which nucleotides comprise heterozygous, homozygous ref, homozygous alt genotypes (i.e. make a call) is up to the user. For any given site, the arrays capture multiple individual nucleotides so one way to do this independent of a reference genome, for a single dataset, is to simply assume that whatever nucleotide is less common is the minor (aka alternate) allele and the other is the major (aka reference) allele. This is an acceptable (and very common) method for analyzing a single dataset but causes obvious problems when trying to compare calls for the same variants between datasets (a nucleotide may have been the alternate in one and reference in the other). Two strategies for making datasets comparable then are:
    1. For each dataset, use knowledge of the probe sequences to determine what strand each nucleotide is on. This appears to be the only completely unambiguous way to ensure that all calls correspond to the same reference and alternate nucleotides.
    2. Use the fact that the SNP arrays at least tell you which nucleotides were measured for each locus to infer, in a fairly quick and dirty way, which strand the probes measured in each dataset. Here are some examples to make this more clear. Let Dataset 1 = D1 and Dataset 2 = D2 in each of these and assume each determine major/minor aka ref/alt alleles based on frequency (where "AC" implies A was designated as the major allele and C as the minor):
        - **Example 1 (AC vs CA)**: D1 says a variant has A as the major allele and C as the minor allele. D2 says C is major and A is minor
            - Correction: For all calls in D2 for this variant, switch the homozygous/heterozygous interpretation (presumably the C allele was more common in D2 but not D1)
        - **Example 2 (AC vs TG)**: D1 says variant has A = major, C = minor and D2 says T = major, G = minor
            - Correction: Nothing for the calls. The probes in this case were for different strands but ultimately captured the same nucleotides (since A is complementary with T and C with G) AND assumed the same major/minor relationship. The allele nucleotides in D2 should be complemented so that the variant is known as AC, but that's it.
        - **Example 3 (AC vs GT)**
            - Correction: As a combination of example 1 and 2, the call interpretation and allele nucleotides should both be swapped in D2 to align with D1.
        - **Example 4 (AT vs TA)**
            - This is where things get tricky. In example 2, we knew that the probes measured different strands simply because A or C nucleotides would be on one strand while T and G nucleotides would be on the other. This definitive knowledge of a strand swap is key. In the AT vs TA case, it could be that the probe measured different strands or it could be that the same strand was used but alleles occurred at different

frequencies in both datasets. We could now say something like, “If the A allele has a frequency of 5% in D1 and it has a frequency of ~5% in D2, then we can safely assume that the same strand was used for the probe”. This, however, becomes problematic as the allele frequencies 50%. The same is true for cases like CG vs GC or even AT vs AT – you simply can’t tell which strand the probes corresponded too without knowledge of the probe seqeuences. These sequences could be compared between the two datasets to determine if they were for the same strand, but they appear to be difficult to come by. This is the main reason why many analyses simply through out A/T and C/G SNPs.

- Here are some helpful discussions/papers on why this step is necessary (and on strand ambiguity in general):
  - StrandScript: evaluation of Illumina genotyping array design and strand correction > Additionally, the strand issue can be resolved by comparing the alleles to a reference genome. Yet, when two alleles of the SNPs are complementary (A/T or C/G), the true strand remains undetermined. The only absolute solution to determine the strand is to compare the probe sequences to a reference genome, providing the probe sequences is correct
  - Genotype harmonizer: automatic strand alignment and format conversion for genotype data integration > However, there are two major challenges to be resolved: … 2) the ambiguous A/T and G/C single nucleotide polymorphisms (SNPs) for which the strand is not obvious. For many statistical analyses, such as meta-analyses of GWAS and genotype imputation, it is vital that the datasets to be used are aligned to the same genomic strand.
  - Is ‘forward’ the same as ‘plus’?… and other adventures in SNP allele nomenclature

bigsnpr/bigstatsr

- In trying to understand how the loadings plots were generated in https://privefl.github.io/bigsnpr/articles/how-to-PCA.html, I found these useful source links:
  - loadings plot function: https://github.com/privefl/bigstatsr/blob/810347dfdbc7b625f0c2907e45e111764c47da8c/R/plot.R#L158
  - big_randomSVD: https://github.com/privefl/bigstatsr/blob/master/R/randomSVD.R#L189
  - what big_randomSVD calls: https://github.com/privefl/bigstatsr/blob/master/R/randomSVD.R#L105
  - where big_SVD class is defined: https://github.com/privefl/bigstatsr/blob/c859ad28d9c6f8c8cc365c3315e3abbb81e128a8/R/SVD.R#L92

GWAS Pipeline Operations

- These are frequent operations in a GWAS workflow:
  - Selection by row/col data or metadata
  - Row/col aggregation
  - Row/col distance and pruning
  - PCA
  - Regression
  - Association testing

Canines

- From Boxer bares all (2005)
  - “All domestic dogs are descended from grey wolves (C. lupus) that were tamed about 15,000 years ago”
  - “Today, researchers published the full genetic code of a 12-year-old boxer named Tasha”
  - The actual publication on the first full genome is Genome sequence, comparative analysis and haplotype structure of the domestic dog.
    - Nature Link
- From Genome sequence, comparative analysis and haplotype structure of the domestic dog
  - On choosing a boxer: “This particular animal was chosen for sequencing because it had the lowest heterozygosity rate among ~120 dogs tested at a limited set of loci; subsequent analysis showed that the genome-wide heterozygosity rate in this boxer is not substantially different from other breeds (Parker 2004)”
    - The “subsequent analysis” is Parker, H. G. et al. Genetic structure of the purebred domestic dog

References