PCA vignette
Principal components analysis with chopsticks

David Clayton

April 14, 2011

Principal components analysis has been widely used in population genetics in order to study population structure in genetically heterogeneous populations. More recently, it has been proposed as a method for dealing with the problem of confounding by population structure in genome-wide association studies.

The maths

Usually, principal components analysis is carried out by calculating the eigenvalues and eigenvectors of the correlation matrix. With N cases and P variables, if we write X for the $N \times P$ matrix which has been standardised so that columns have zero mean and unit standard deviation, we find the eigenvalues and eigenvectors of the $P \times P$ matrix $X^{\mathrm{T}}.X$, The first eigenvector gives the loadings of each variable in the first principal component, the second eigenvector gives the loadings in the second component, and so on. Write the first C component loadings as columns of the $P \times C$ matrix B. Then, to calculate the principal component scores for subjects, S, we would simply apply the factor loadings to the original data, *i.e.* $S = X.B$.

This standard method is rarely feasible for genome-wide data since P is very large indeed and calculating the eigenvectors of $X^{\mathrm{T}}.X$ becomes impossibly onerous. However, the calculations can also be carried out by calculating the eigenvalues and eigenvectors of the $N \times N$ matrix $X.X^{\mathrm{T}}$. Then the eigenvectors give the principal component scores, S, and the loadings are calculated by $B = X^{\mathrm{T}}.S.D^{-1/2}$, where D is a $C \times C$ diagonal matrix containing the first C eigenvalues. Using this method of calculation, it is only (!) necessary to find the eigenvalues and eigenvectors of an $N \times N$ matrix. Current microarray-based genotyping studies are such that N is typically a few thousands and P is a few tens of thousands.

An example

In this exercise, we shall calculate principal component loadings in controls alone and then apply these loading to the whole data. This is more complicated than the simpler procedure of

calculating principal components in the entire dataset but avoids component loadings which unduly reflect case/control differences; using such components to correct for population structure would seriously reduce the power to detect association since one would, to some extent, be "correcting" for case/control differences[1]. We will also "thin" the data by taking only every tenth SNP. We do this mainly to reduce computation time but thinning is often employed to minimize the impact of linkage disequilibrium (LD), to reduce the risk that the larger components may simply reflect unusually long stretches of LD rather than population structure. Of course, this would require a more sophisticated approach to thinning than that used in this demonstration.

In a more sophisticated approach, one might use the output of `snp.imputation` to eliminate all but one of a groups of SNPs in strong LD for thinning.

We shall use the data introduced in the main vignette. We shall first load the data and extract the controls.

```
> require(chopsticks)
> data(for.exercise)
> controls <- rownames(subject.support)[subject.support$cc == 0]
> use <- seq(1, ncol(snps.10), 10)
> ctl.10 <- snps.10[controls, use]
```

The next step is to standardize the data to zero mean and unit standard deviation and to calculate the $X.X^{\mathrm{T}}$ matrix. These operations are carried out using the function `xxt`.

```
> xxmat <- xxt(ctl.10, correct.for.missing = FALSE)
```

The argument `correct.for.missing=FALSE` selects a very simple missing data treatment, *i.e.* replacing missing values by their mean. This is quite adequate when the proportion of missing data is low. The default method is more sophisticated but introduces complications later so we will keep it simple.

When performing a genome-wide analysis, it will usually be the case that all the data cannot be stored in a single `snp.matrix` object. Usually they will be organized with one matrix for each chromosome. In these cases, it is straightforward to write a script which carries out the above calculations for each chromosome in turn, saving the resultant matrix to disk each time. When all chromosomes have been processed, the $X.X^{\mathrm{T}}$ matrices are read and added together.

The next step of the calculations requires us to calculate the eigenvalues and eigenvectors of the $X.X^{\mathrm{T}}$ matrix. This can be carried out using a standard R function. We will save the first five components.

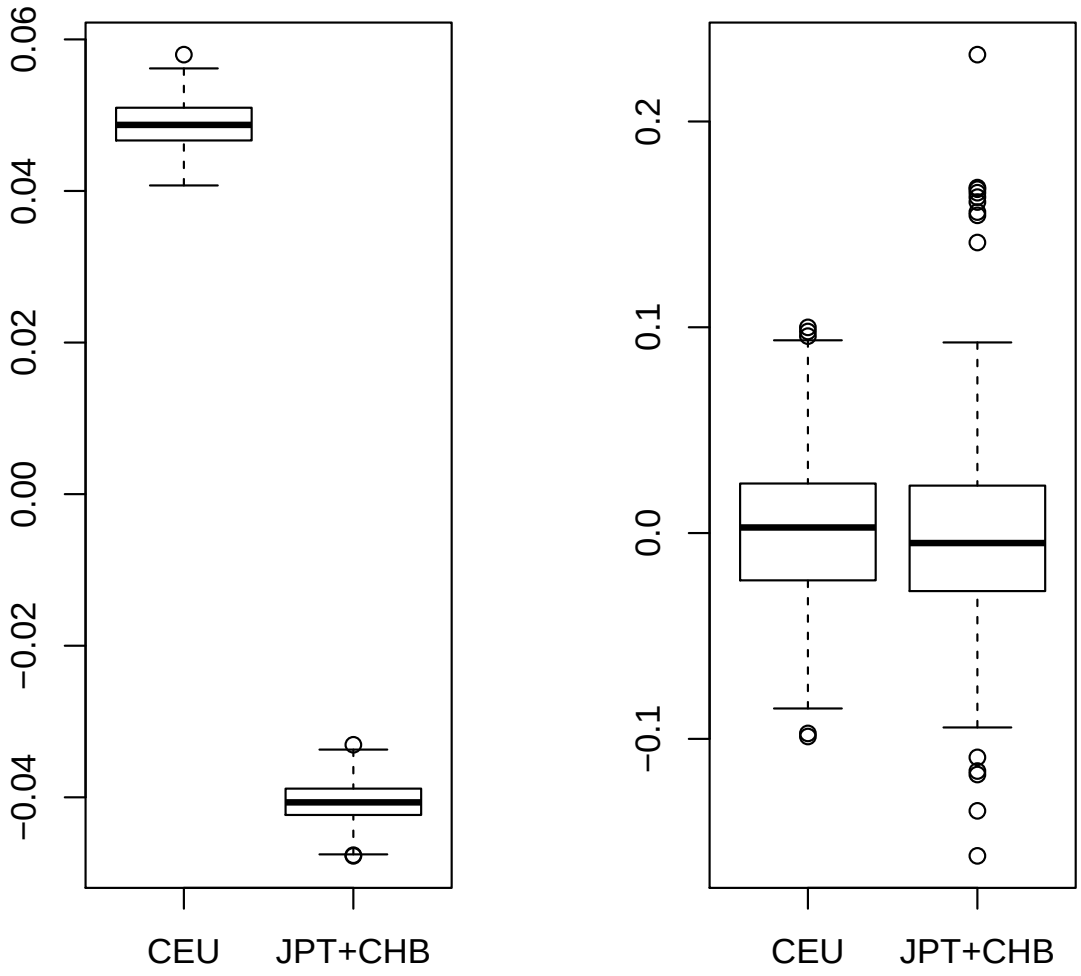
```
> evv <- eigen(xxmat, symmetric = TRUE)
> pcs <- evv$vectors[, 1:5]
> evals <- evv$values[1:5]
> evals
```

[1] An alternative approach is to standardise the X matrix so that each column has zero mean in both cases and controls. This can be achieved by using the `strata` argument in the call to `xxt`. Here, however, we have used controls only since this reduces the size of the matrix for the eigenvalue and vector calculations.

```
[1] 167181.398   8890.117   8709.265   8503.632   8379.124
```

The first principal component has a markedly larger eigenvalue and we might hope that this reflects population structure. In fact these data were drawn from two very different populations, as indicated by the `stratum` variable in the subject support frame. The next set of commands extract this variable for the controls and plot box plots for the first two components by stratum.

```
> pop <- subject.support[controls, "stratum"]
> par(mfrow = c(1, 2))
> boxplot(pcs[, 1] ~ pop)
> boxplot(pcs[, 2] ~ pop)
```



Clearly the first component has captured the difference between the populations. Equally clearly, the second principal component has not.

The next step in the calculation is to obtain the SNP loadings in the components. This requires calculation of $B = X^{\mathrm{T}}.S.D^{-1/2}$. Here we calculate the transpose of this matrix, $B^{\mathrm{T}} = D^{-1/2}S^{\mathrm{T}}.X$, using the special function `snp.pre.multiply` which pre-multiplies a `snp.matrix` object by a matrix after first standardizing it to zero mean and unit standard deviation.

```
> btr <- snp.pre.multiply(ctl.10, diag(1/sqrt(evals)) %*% t(pcs))
```

We can now apply these loadings back to the entire dataset (cases as well as controls) to

derive scores that we can use to correct for population structure. To do that we use the function `snp.post.multiply` which post-multiplies a `snp.matrix` by a general matrix, after first standardizing the columns of the `snp.matrix` to zero mean and unit standard deviation. Note that it is first necessary to select out those SNPs that we have actually used in the calculation of components.

```
> pcs <- snp.post.multiply(snps.10[, use], t(btr))
```

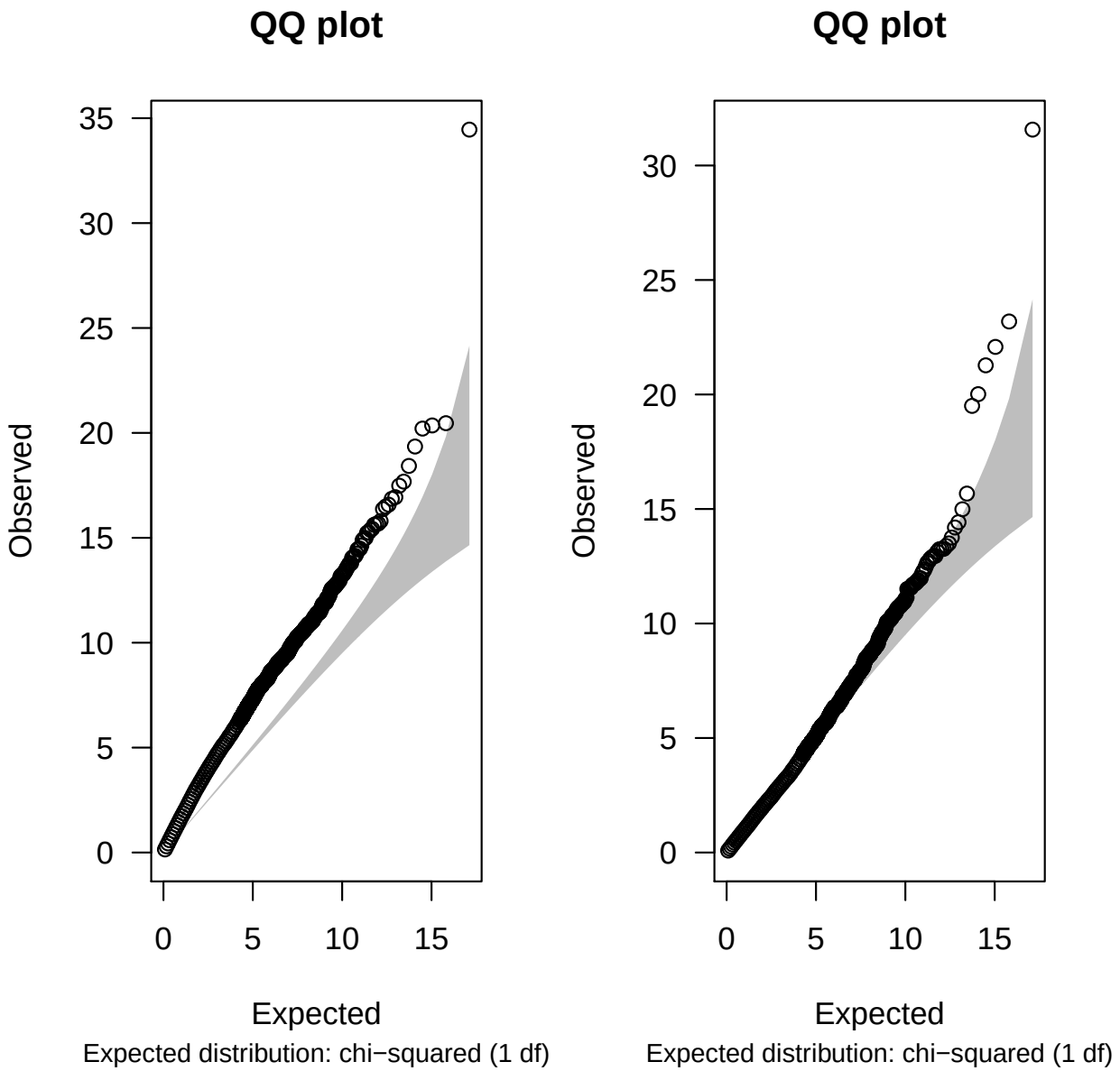
Finally we shall evaluate how successful the first principal component is in correcting for population structure effects. (`snp.rhs.tests` return `glm` objects.)

```
> cc <- subject.support$cc
> uncorrected <- single.snp.tests(cc, snp.data = snps.10)
> corrected <- snp.rhs.tests(cc ~ pcs[, 1], snp.data = snps.10)
> par(mfrow = c(1, 2), cex.sub = 0.85)
> qq.chisq(chi.squared(uncorrected, 1), df = 1)

           N      omitted       lambda
28497.000000     0.000000     1.712338

> qq.chisq(chi.squared(corrected), df = 1)

           N      omitted       lambda
28497.000000     0.000000     1.008964
```



The use of the first principal component as a covariate has been quite successful in reducing the serious over-dispersion due to population structure. Indeed it is just as successful as stratification by the observed `stratum` variable.