Generating Simulated SNP Array Data

**Simulating the Tumor Genome**

First, before we can generate simulated data, we must simulate a ‘truth.’ Specifically, we must simulate a tumor, or a tumor genome. Given that we are not thinking of sequencing data we don’t need to simulate the tumor with a granularity at the level of individual bases; rather, we are only interested in simulating ‘true’ copy number states across the genome at the level of genomic ‘segments’, where a segment is defined as a contiguous set of markers (of which there are about 600,000 in the entire genome for this array, or an average of about 20-30,000 per chromosome). Because we want our simulated data to resemble real data as much as possible, rather than simulating marker positions across the genome, we use the marker positions from a real SNP array. This way, the density of markers (or variation in the density of markers) across the genome recapitulates what one might expect to see in actual data.

Next, we divide the genome into segments. We are assuming that, as far as copy number alterations are concerned, genomes can be characterized as consisting of segments varying in length (up to the size of the length of the chromosome on which the segment lay, of course). Each segment has a copy number value associated with it, where segments that are normal have two copies present (we are ignoring the sex chromosomes for now). Again, in order to generate realistic simulations, we have used the number of segments imputed in actual patient samples (usually between 100 and 400 segments in a typical CLL sample’s genome) to determine how many ‘pieces’ to divide the genome into.

Another important point worth noting is that, although if all SNP markers were located at points of heterogeneity we should expect to see one copy of each allele, this is not in fact the case. Many markers (even in normal patient samples) are at homogeneous loci, i.e., there are two copies of one allele. For this reason, in order to make the simulations realistic, we randomly select some fraction of the markers (perhaps a third or half) to be made ‘homogeneous’, meaning that in the normal case, there are two copies of one allele rather than one copy each of two alleles.

So, at this point in the simulation process what we have is a data frame consisting of SNP locations (locations on the chromosome, that is), chromosome numbers (the chromosome on which each SNP is found), a ‘segment number’ (e.g., 2.5 would mean the 5th segment on chromosome 5), number of A-copies and number of B-copies (given that we are still describing a normal genome, A + B equals 2 for all SNPs). From here, we generate the aberrant subclones. This is done by randomly selecting a segment (or multiple contiguous segments) to be assigned a new (non-normal, with equal probability of loss and gain of copy) copy number value. It should be noted that with each ‘iteration’ (really, each new clone generated), the new clone can be generated either by creating an aberration from the normal genome, or by adding new copy number alterations to an existing subclones, so if there are more than two subclones, one subclone can be a direct descendant of another. Also, it should be noted that one can generate a large population of clones, then select only some of them to be used to comprise the simulated ‘tumor’, so as to simulate the effect of clones dying off.

The last element of the ‘truth’ to be simulated is the ‘psi’ vector, which denotes the fraction of tumor cells belonging to each clone. The ‘psi’ values are generated via a K-dimensional Dirichlet distribution (where K is the number of subclones of which the tumor is comprised) with alpha parameter 1.

**Adding Noise**

To make simulated data from a simulated truth, we must add noise. For simplicity’s sake, we first generate what might be called ‘pre-data’ or ‘noiseless data.’ Or, to put it another way, we compute the weighted average allelic copy number across all clones at each marker. Essentially, for each SNP, we take all K ‘A-values’, multiply them by the associated ‘psi values’ and take the sum, then do the same with the B-allele copy number values at each SNP. So, if there are two clones, with 1 copy of A and 2 copies of A, respectively, at a particular allele, while psi1 = 1/3 and psi2 = 2/3, the weighted average A-allelic copy number is 1.667.

Next, we add noise to the weighted average copy number values for each marker. Currently, the noise is generated from a normal distribution with a predetermined value for sigma (usually ~1). This poses the obvious issue that, in theory, the addition of noise could lead to an observed negative copy number value. Since the variance used in these simulations is small enough that only a minute fraction (if any) of SNPs end up with negative copy number values after noise addition, it can be easily dealt with ad hoc without significantly affecting the distribution of the ‘observed’ copy number values.

**Converting Allelic Copy Number into Log R Ration and B Allele Frequency**

Next, we compute the Log R Ratio and B Allele Frequency at each SNP from the noise-added copy number values. The LRR and BAF are what is actually observed in the real data. Finally, for the purpose of analysis, we compute the segment median LRR and BAF for each segment (while also recording the length of each segment), which is also done for actual patient samples.

[Compare plot of actual SNP array data with simulated SNP array data]

**Population of Samples to Generate**

-Population characteristics

The simulated tumors possess anywhere from 1 to 6 (?) subclones, with the fraction of cells belonging to each clone determined by a dirichlet distribution.

**Parameter Selection**

-sigma

-theta

-…

**Assessing Accuracy**

There are several ways to assess the accuracy of the imputed clonal architecture of the tumor. The most basic way is to compute the absolute accuracy of clonal copy number assignments across all genomic segments. Essentially, the metric would equal the total number of accurate copy number assignments across the ‘metagenome’ divided by the total number of assignments to be made (which equals the product of the number of segments, the number of subclones, and the number of possible alleles, which we assume to be 2). This metric is the strictest metric. It suffers from the shortcoming that it does not measure the severity of an assignment error. For example, if clone A has 0 copies of an allele at a given segment, and an algorithm imputes 1 copy, this error is less severe than if it were to impute 4 copies. The absolute accuracy metric considers both errors to be equal.

Another problem with the simple version of the absolute accuracy metric is that it can be misleading if not considered in the context of the right ‘null hypothesis.’ For example, even for most cancer genomes, if one assumes the entire genome to be normal in terms of copy number, one will still consistently get as much as 80-90% or higher of the segments accurate. In this light, an accuracy rate of 80%, though superficially much better than one would expect ‘by chance’ is not necessarily particularly impressive.

Another way to measure accuracy is the sum of the R^2 values. This metric (or pair of metrics, rather) avoids the disadvantage of treating all errors as though they were equally ‘erroneous.’ However, it has the disadvantage in that it cannot be reported as a single metric gauging the overall performance of an algorithm, because one must report the sum of the ‘psi’ R^2 values and the sum of the ‘copy number’ R^2 values separately.

Probably the most robust way to assess performance is the computation of a p-value for the parameter set. Essentially,

-How to account for (possible) correlation of importance with size of subclone

A significant shortcoming of the above mentioned performance metrics is that each essentially weights assignment errors form all clones equally. There is reason to consider an assignment error for a large clone to be more ‘severe’ than an error for a small clone. For this reason, we introduce yet another accuracy metric…

-Accuracy in cases where K is incorrectly inferred.

**Algorithm Testing Strategy**

Algorithm classes:

-EM

-MCMC (segment-wise?)

-Genetic Alg.

Algorithm Types:

-AB-model

-CNV state model

**Assessing Accuracy over Variations in Parameter Values**

**Code Structure**