FG Assignment 2

USN: 303039534

2 Simulating Copy Number Profiles

We start by writing a function cnv_sim(N) that simulates N copy number changes from an initially homogenous human genome using the function cn_change to generate each copy number change. We initiate each change uniformly at random along the genome. Since a higher copy number implies more of a particular DNA sequence, we need to weigh our sampling by the copy number. This also ensures that once a sequence has a copy number of zero, the loss of information is irreversible since in this case we are multiplying our probabilities by zero. The probabilty of a copy number change occuring in segment i is thus given by

$$p_i = \frac{L_i \cdot cn_i}{\sum_j L_j \cdot cn_j}$$

Here, L_i and cn_i is the length and copy number of segment i respectively. After picking a segment, we pick the initiation site uniformly at random within the segment. The length of a copy number change is drawn uniformly at random from a distribution ranging from 1 to 100 Mb. We let the change extend symmetrically around the initation site but bounded by the beginning and end of the segment in which it occurs. This implementation is roughly equivalent to assuming that a break is equally likely between any two bases in any chromosome, but with a bias towards changes extending to the end of a segment and towards smaller segments due to the hard segment cut-off. Real biological systems are more complex than this, but it is sufficient for the purpose of verifying our algorithm in sections 7 and 8. The simulation is run on a dataframe storing the chromosome, length and copy number of each segment. Segments are stored in order along the genome which allows us to reconstruct the full genome from our segments. For all of our simulations, we assume the samples to be female to avoid complications in assigning initial X and Y copy numbers for multiploid genomes. However, the code can easily be extended to also initialize Y chromosomes.

```r
set.seed(26110940) #set seed for repeatable simulations
chr_size_file = 'hg19.chrom.sizes.txt' #change if different file name

cn_change = function(results, tot.size, n_change){
  #simulates a single copy number change given a data frame of segments (results)
  #a cn change is initiated uniformly at random across the genome
  #it then extends symmetrically in both directions with a length
  #of 1-100 Mb picked uniformly at random, bounded by the segment boundaries
  #tot.size is the total genome size in basepairs, n_change the max cn change

  #probability of initating a cn change at each basepair
  #is proportional to its length and current copy number
  ps = cumsum( c(0, results[,'length']*results[,'cn']) / tot.size )
  pos = runif(1)
  #first find segment in which a CN change occurs, then the point of change
  seg = sum (as.numeric(pos > ps)); chr = as.character(results[seg, 'chr']);
  L = results[seg, 'length']

  site = round(runif(1)*L,0); size = round(runif(1, 10^6, 10^8), 0)
  init = max(1, site-floor(size/2)) #cn change cannot go past segment start
  fin = min(L, site+ceiling(size/2)) #cn change cannot go past segment end

  change = sample( c( -n_change:-1 , 1:n_change ), 1)
  new_cn = results[seg,'cn']+change #old copy number plus change
  new_cn = max(0, new_cn) # don't allow negative copy numbers
  tot.size = tot.size + change*(fin-init) #update our total amount of DNA

  if ( isTRUE(all.equal( c(init,fin), c(1,L) ) )){ #complete overlap, update cn
    results[seg,'cn']=new_cn
  }else if (init == 1){ #creates new CN at start of segment
    results[seg,'length'] = L - fin
    if(seg==1){results = rbind(list(chr, fin, new_cn), results)
    }else{results = rbind(results[1:(seg-1),],
                          list(chr, fin, new_cn), results[-(1:(seg-1)),])}
  }else if (fin == L){ #new CN at end of segment
    results[seg,'length'] = init
    results = rbind(results[1:seg,], list(chr, L-init, new_cn), results[-(1:seg),])
  }else{ #cn change falls within segment
    if(seg == 1){results = rbind(results[1,], list(chr, size, new_cn), results)
    }else{results = rbind(results[1:seg,],
                          list(chr, size, new_cn), results[-(1:(seg-1)),])}
    results[seg,'length']= init
    results[seg+2,'length']= L-fin
  }
  return(list(results, tot.size))
}

cnv_sim = function(N, n_change = 1, noise = 0, ploidy=2){
  #simulates N copy number changes in a hypothetical human genome
  #n_change is max copy number change in a single event
  #noise specifies whether we add Gaussian noise, ploidy specifies initial ploidy
  #we assume samples to be female
  results = read.table(chr_size_file, stringsAsFactors = 0)
  results[,3] = rep(ploidy,24) #initialize genome
  colnames(results) = c('chr', 'length', 'cn')
  results[results[,'chr'] == 'chrY', 'cn'] = 0 #assume female
  tot.size = ploidy*3036303846 #total number of base pairs
  for (i in 1:N){
    new_res = cn_change(results, tot.size, n_change) #simulate cn change
    results = new_res[[1]]; tot.size = new_res[[2]]
  }
  colnames(results)=c('chr', 'length', 'cn'); rownames(results)=1:dim(results)[1]
  #Add Gaussian noise to all segments if specified
  if (noise){ results[,3] = results[,3] + rnorm(dim(results)[1], sd=0.1) }
  return(results)
}
```

The first 10 segments of an example simulation of 50 copy number changes are given in table 1.

```
res = cnv_sim(50, n_change = 1, noise = 0, ploidy=2)
xres = xtable(res[1:10,],
              caption = 'First 10 segments following
              a simulation of 50 copy number changes',
              digits=c(0,0,0,0), label='tab:sim')
```

```
addtorow = list()
addtorow$pos = list(0)
addtorow$command <- c(' Segment & Chromosome & Length & Copy Number \\\\\n' )
align(xres) <- rep("r", 4)
print(xres, include.colnames = 0, add.to.row = addtorow)
```

Segment	Chromosome	Length	Copy Number
1	chr1	70934013	2
2	chr1	22654494	3
3	chr1	93777278	2
4	chr1	27836603	1
5	chr1	9107794	2
6	chr1	24940439	2
7	chr2	20398012	2
8	chr2	11119108	3
9	chr2	10134647	3
10	chr2	116703974	2

Table 1: First 10 segments following a simulation of 50 copy number changes

We also write a function plot_results() to visualize the resulting genomic profile and a function sim_more_cnvs() to simulate multiple copy number changes and plot the resulting profiles.

```
plot_results = function(results, ploidy='NA', n_change='NA', noise=2, main=''){
  #given a dataframe of segment lengths and cns, plots copy number profile
  cum = cumsum(results[,2]) #x boundaries
  ymax = min( max(results[,3], na.rm=T)+1, median(results[,3], na.rm=1)+5)
  plot(c(1, 3095677412), c(2,2), type='n', lty=2,
       xlim = c(1, 3095677412), ylim = c(-0.5, ymax),
       xlab = '', ylab = '', main = main, xaxt = 'n'
       ) #initialize plot

  segments( 1, seq(0, floor(ymax), 1), #plot integer lines
            x1 = 3095677412, y1 = seq(0, floor(ymax), 1), lwd=0.2, lty=2  )
  segments( c(0, cum[-length(cum)]), results[, 3], #plot copy numbers
            x1 = cum, col='blue', y1 = results[, 3], lwd=3  )

  chr_sizes = read.table(chr_size_file)
  cum = cumsum(as.numeric(chr_sizes[,2])); cum = cum[-length(cum)]
  segments( cum, -0.5, x1 = cum, ymax, lwd=0.5) #plot chromosome delineations
}

sim_more_cnvs = function(cnvs = c(3,10,25,100,200), n_change=1, noise=0, ploidy=2){
  #function for simulating multiple CNVs and putting in a plot
  par(mfrow=c(length(cnvs),1), mar=c(0.5,2.5,0.5,1), xaxs='i', yaxs='i', lend=1)
  results = vector('list', 0)
  for (cnv in cnvs){
    results[[cnv]] = cnv_sim(cnv, n_change, noise, ploidy) #simulate changes
    plot_results(results[[cnv]], ploidy, n_change, noise) #plot result
  }
  return(results)
}
```

Using these functions, we simulate and plot genomic profiles with 3, 10, 25, 100 and 200 copy number changes (figure 1) from a diploid genome. In this and all subsequent plots, the x-axis indicates genomic posision with chromosomes ordered from 1:22 followed by the X and Y chromsomes. The beginning and end of each chromosome is indicated by a vertical line.

```
res1 = sim_more_cnvs( noise=FALSE, n_change=1)
```

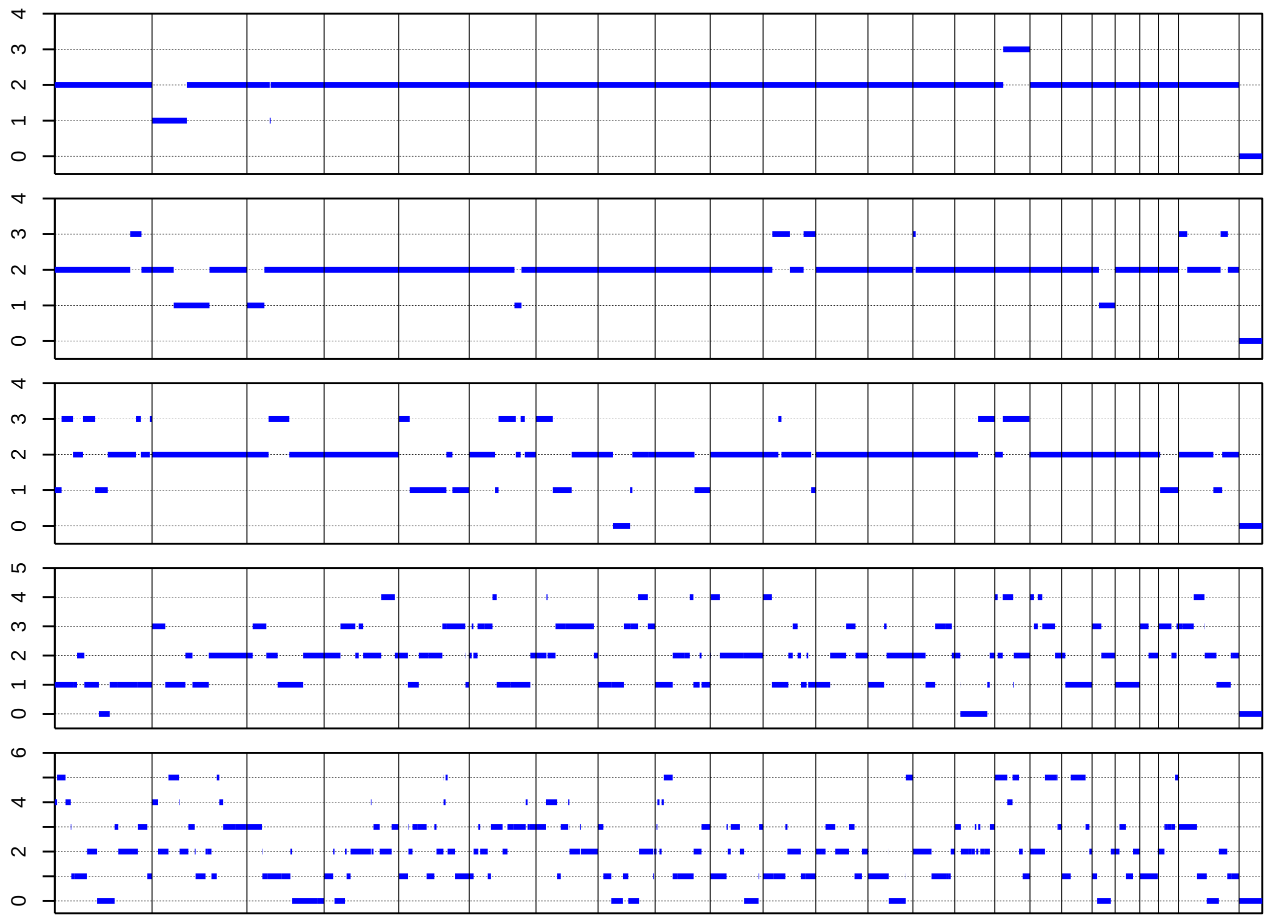


Figure 1: Simulated copy number profiles after 3 (top), 10, 25, 100 and 200 (bottom) copy number changes. The y axis indicates absolute copy number, the x axis indicates position in the human genome. Vertical lines indicate chromosome boundaries with chromosomes ordered from 1:22 followed by the X and Y chromosomes.

3 Introducing Noise

We can simulate segmentation noise with our sim_cnv() function by introducing Gaussian noise with a standard deviation of 0.1 to all segments. We can also allow for copy number changes of more than one copy per change. This is specified by providing the optional argument n_change to sim_cnv(). In this case, copy number changes are drawn randomly from positive or negative integer values ranging from 1 to n_change without allowing changes that would lead to negative copy numbers. Copy number profiles for simulations of 3, 10, 25, 100 and 200 copy number changes with Gaussian noise and n_change = 2 are given in figure 2.

```
res2 = sim_more_cnvs( noise=TRUE, n_change=2)
```

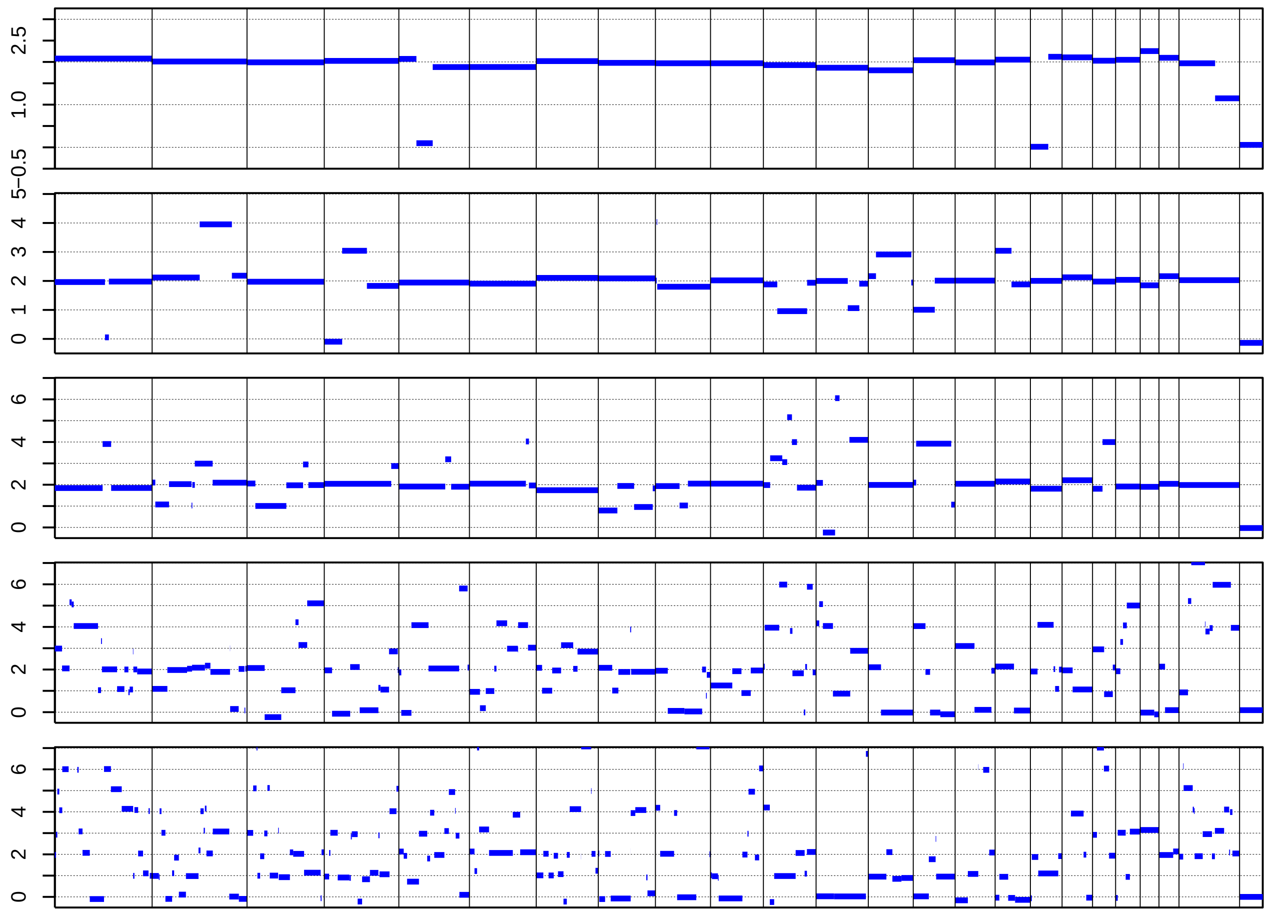


Figure 2: Copy number profiles for simulations including segmentation noise and n_change = 2. The number of copy number changes for the profiles are 3 (top), 10, 25, 100 and 200 (bottom).

4 Altering Background Ploidy

Many tumor cells have undergone full genome duplications, and we may thus want to simulate copy number changes from a reference genome that is not diploid. We achieve this by introducing the optional argument 'ploidy' with a default value of 2, specifying the ploidy of our reference genome. Setting ploidy=4, we simulate copy number changes from a tetraploid genome with the resulting genomic profiles given in figure 3.

```
res3 = sim_more_cnvs( noise=TRUE, n_change=2, ploidy=4)
```

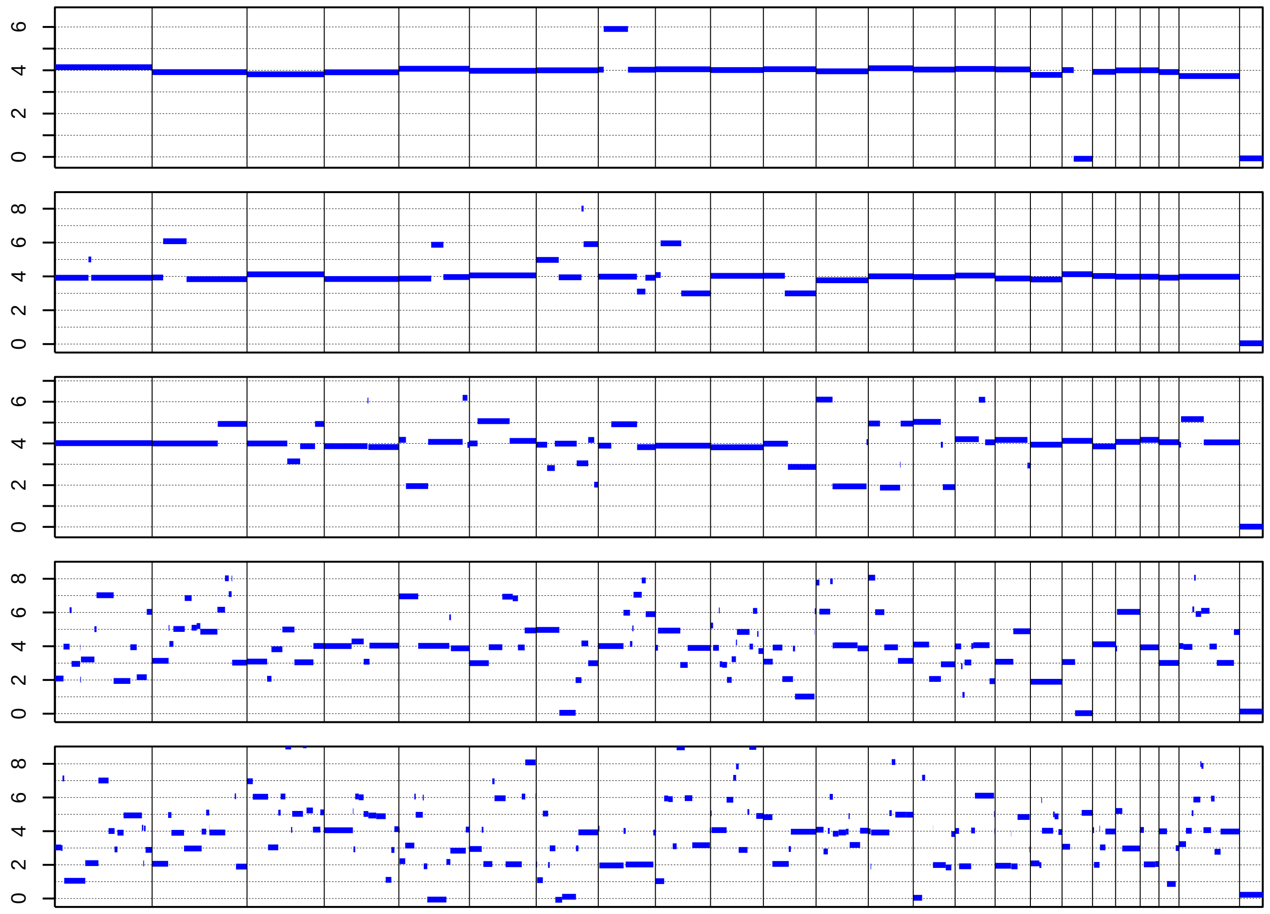


Figure 3: Copy number profiles for copy number changes from a tetraploid chromosome, including segmentation noise and with n_change = 2. The number of copy number changes for the profiles are 3 (top), 10, 25, 100 and 200 (bottom).

5 Normalization and Assessing Clonal Similarity

We proceed to write a function normalize() to normalize these absolute copy number values by the median copy number value to simulate the relative copy number data we get from experiments. We also write a function clonal_proximity() that assesses how close a given copy number profile is to being clonal (i.e. to having integer copy numbers). This is quantified using a Euclidean clonal distance metric weighted by segment length:

$$d = \sqrt{(\sum_i cn_i - \text{round}(cn_i, 0))^2 \cdot L_i) \cdot \frac{4}{L}}$$

I.e. we consider how far the copy number of each segment is from being an integer, weigh this by the segment length, and normalize by the total genome size L. Weighting by segment length ensures that each position in the genome contributes equally to the distance metric, and using a Euclidean distance was found to yield better results than a linear (Manhattan) or higher power distance. This clonal distance is 0 if all segments have an integer copy number and 1 if all segments have half-integer copy numbers. We therefore define a normalized clonal similarity metric as $sim = 1 - d$ which also runs from 0 to 1. Finally, we write a function test_proximity() that takes as input a dataframe of relative copy number data for a set of segments, scales the copy numbers linearly, and finds the scaling factor that optimizes the clonal similarity metric. This gives an estimate of absolute copy numbers.

```r
normalize = function(results){
  #performs median normalization
  med = median(results[,'cn']); results[,'cn']=results[,'cn']/med
  cat('normalized data by', round(med, 2))
  return(results)}

clonal_proximity = function(results, n=2){
  #quantifies how close a copy number profile is to being clonal
  #we use a Euclidean distance normalized by segment length
  #proximity is 1 if all segments have integer cn, 0 if they are all half-integer

  L = sum(results[,'length'])
  d = ( sum( (results[,'cn'] - round(results[,'cn'],0) )^n
          * results[,'length'] )*n^2/L )^(1/n)
  sim = 1-d; #similarity is one minus normalized distance
  return(sim)
}

test_proximity = function( results, scale_factors = 'default', main='default',
                           plot_relative = 0){
  #given a dataframe of relative copy numbers (results)
  #finds a scaling to convert it to absolute copy number using the
  #clonal_proximity() function. Plots relative and absolute copy number profiles
  #together with the clonal similarity as a function of scaling

  res = results[ !is.na(results[,'cn']), ] #only consider regions with cn data
  sims = c()

  if (scale_factors == 'default'){
    scale_factors = seq( 0.5, 6, l=1101 ) #go from .5ploid to hexaaploid reference
  }
  test_res=res
  for (scale in scale_factors){
    test_res[,'cn'] = res[,'cn']*scale #scale copy numbers
    sims = c(sims, clonal_proximity(test_res)) #assess the quality of our result
  }

  if (plot_relative){#plot relative cn profile
    par(mfrow=c(3,1),mar=c(0.5,3.5,1.5,1),xaxs='i',yaxs='i',lend=1,cex.main=1.1)
    plot_results(results, main = paste('Relative Copy Number Profile', main) )
  }else{par(mfrow = c(2,1), mar = c(0.5,3.5,1.5,1),
          xaxs = 'i', yaxs = 'i', lend=1, cex.main = 0.8)}

  simmax = max(sims); scalemax = scale_factors[which(sims==simmax)]
  results[,'cn'] = results[,'cn']*scalemax[1] #calculate absolute profile

  #plot aboslute profile
  plot_results(results, main = paste('Inferred Clonal Profile', main) )

  par(mar = c(3.5,3.5,1.5,1)) #plot clonal similarity vs scaling
  plot(scale_factors, sims, type='l', main=paste('Scaling Profile', main),
       xlab = '', ylab = '', ylim = c(0,1), yaxt='n')
  axis(2, at = seq(0, 1, length = 2), labels=c(0,1) ,srt=45,tick=TRUE)
  title(ylab="clonal similarity", line=2, cex.lab=0.9)
  title(xlab="scaling factor", line=2, cex.lab=0.9)

  cat('scaling:', scalemax, 'clonal similarity:', simmax)
  return(list('results'=results, 'scaling'=scalemax, 'clonal'=simmax))
}
```

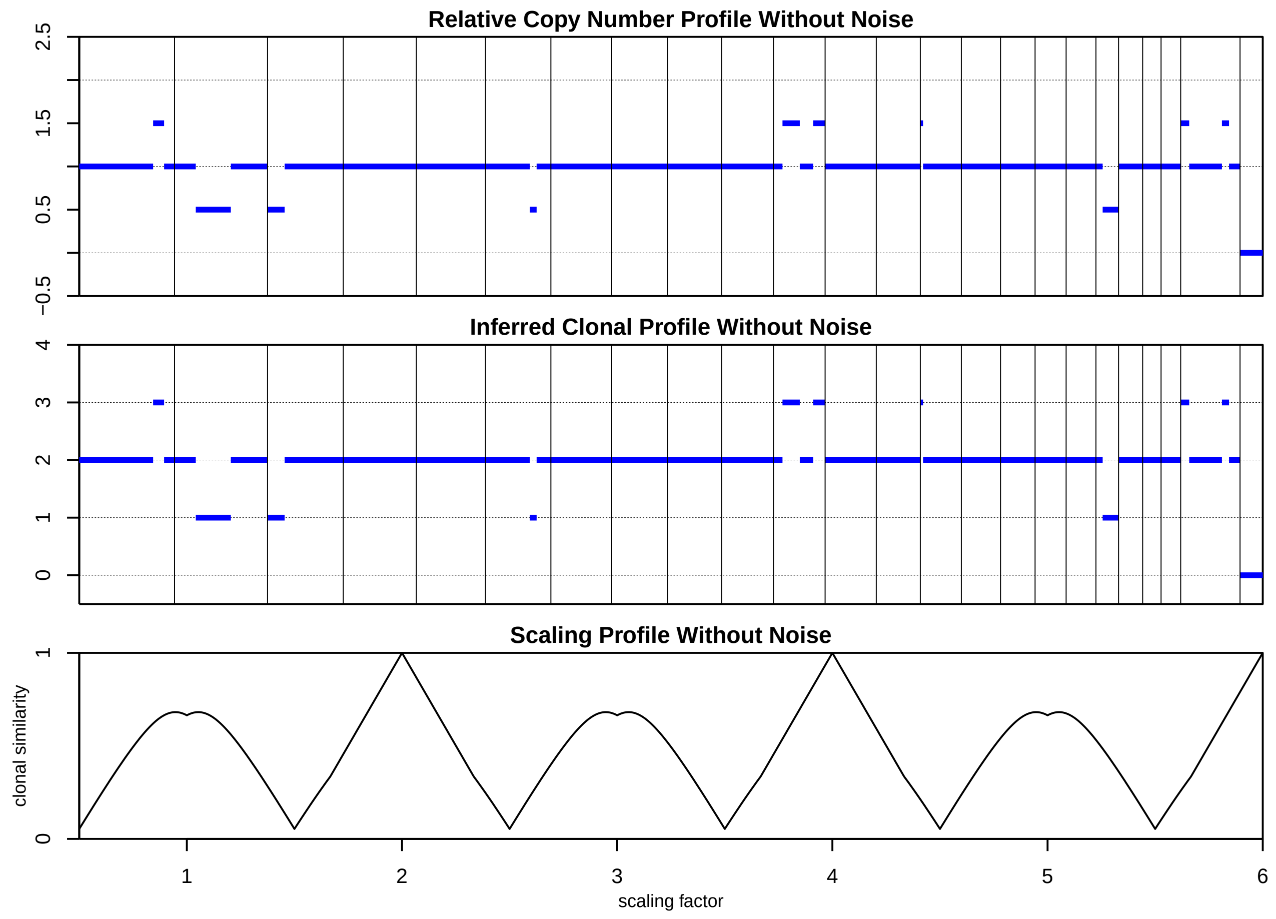
6-7 Generating Relative Copy Numbers and Transforming to Absolute

To assess the utility of these functions for estimating absolute copy numbers, we generate a relative copy number profile from a diploid reference with 10 copy number changes, no noise, and a maximum copy number change of 1. Figure 4 shows the original relative profile, the inferred absolute copy number profile, and how the clonal similarity varies with scaling.

```
res1_norm = normalize(res1[[10]])

## normalized data by 2

shift1 = test_proximity(res1_norm, main = 'Without Noise', plot_relative=1)
```



```
## scaling: 2 4 6 clonal similarity: 1
```

Figure 4: Relative (top) and inferred absolute (middle) copy number profiles for a simulation of 10 single copy number changes. Bottom: clonal similarity as a function of scaling. The inferred scaling factor is correctly 2, but with multiples of 2 giving equally good fits.

The clonal similarity profile is in this case periodic with a period of 2, which is a consequence of the fact there there is no noise in the underlying data.

8 Converting Noisy and Multiploid Relative Copy Numbers to Absolute

We see that in the case with no noise, scaling factors of 2, 4, 6 etc. provide equally good clonal profiles. However, we can resolve this degeneracy in the case of noisy data. In this case we can write

$$R = \alpha S + \beta$$

where R is our relative signal, S is the true signal and β is noise. This implies

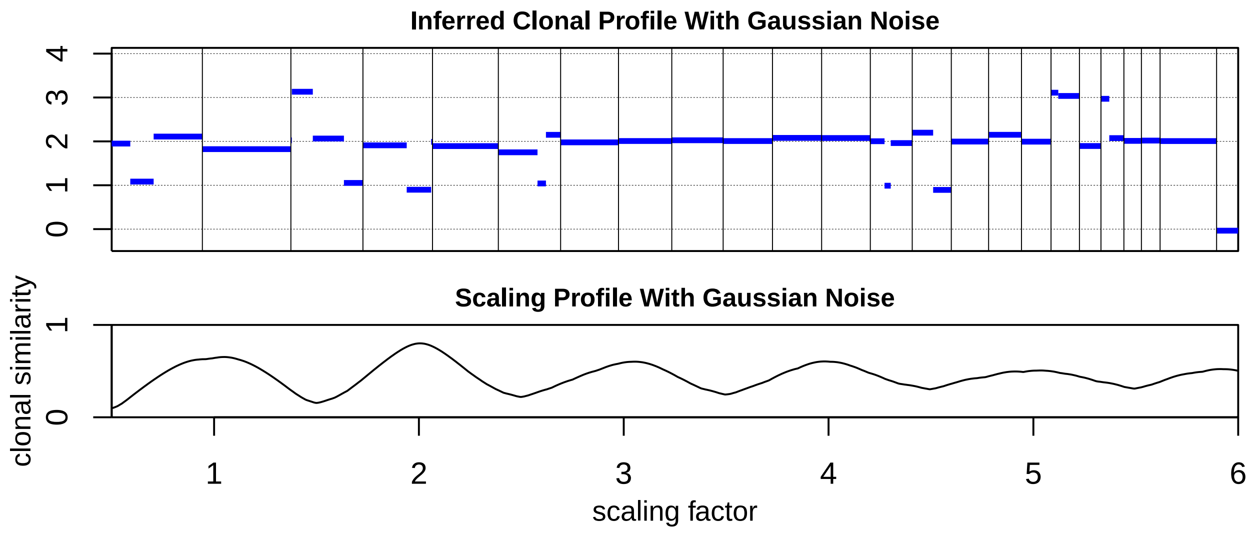
$$S = \frac{R}{\alpha} - \frac{\beta}{\alpha}$$

Now setting $\frac{1}{\tilde{\alpha}} = 2\frac{1}{\alpha}$ we double the noise term leading to a worse fit to a clonal profile. This is exemplified in figure 5 which is similar to figure 4 except that it includes Gaussian noise and we no longer show the relative copy number profile. We see that in this case, scaling by 2 gives a uniquely good clonal similarity since a scaling of 4 increases the noise term.

```
res4 = cnv_sim (10, n_change = 1, noise = 1, ploidy=2)
res4_norm = normalize(res4)

## normalized data by 1.99

shift4 = test_proximity(res4_norm, main = 'With Gaussian Noise')
```



```
## scaling: 2.005 clonal similarity: 0.800448
```

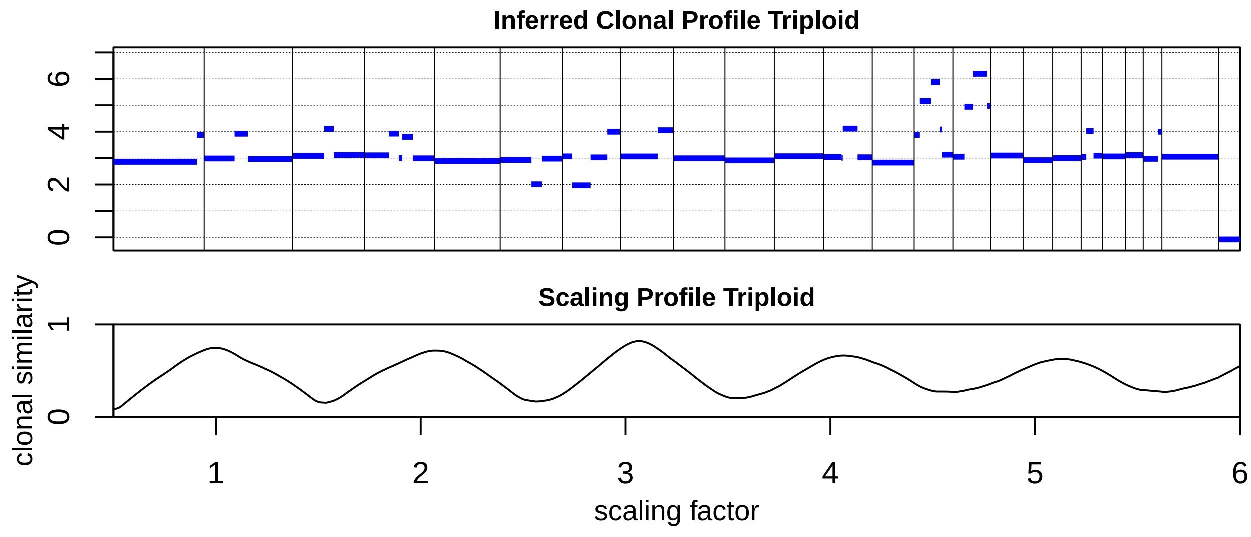
Figure 5: Top: inferred absolute copy number profile for a simulation of 10 single copy number changes with Gaussian noise. Bottom: clonal similarity as a function of scaling. The inferred scaling factor is unique and correctly has a value of approximately 2.

Given the above considerations, we may also ask whether our proposed procedure can be used to infer correct absolute copy numbers when the reference genome is not diploid or when more copy number changes have occurred. In figure 6 we therefore perform a similar analysis generating 20 copy number changes from a triploid reference genome and in figure 7 50 copy number changes from a tetraploid reference genome allowing for changes of up to two copies per copy number change.

```
res5 = cnv_sim (20, n_change = 1, noise = 1, ploidy=3)
res5_norm = normalize(res5)

## normalized data by 3.06

shift5 = test_proximity(res5_norm, main = 'Triploid')
```



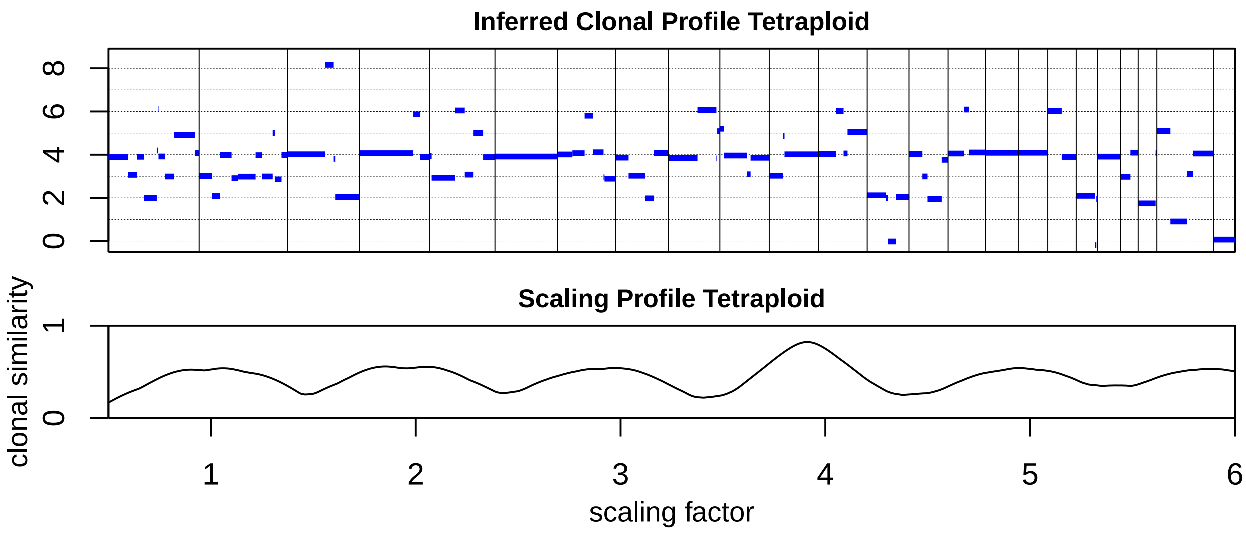
```
## scaling: 3.065 clonal similarity: 0.8195685
```

Figure 6: Top: inferred absolute copy number profile for a simulation of 20 single copy number changes with Gaussian noise and a triploid reference genome. Bottom: clonal similarity as a function of scaling. The inferred scaling factor is unique and correctly has a value of approximately 3, reflecting the triploid reference genome.

```
res6 = cnv_sim (50, n_change = 2, noise = 1, ploidy=4)
res6_norm = normalize(res6)

## normalized data by 3.91

shift6 = test_proximity(res6_norm, main = 'Tetraploid')
```



```
## scaling: 3.91 clonal similarity: 0.8231187
```

Figure 7: Top: Inferred absolute copy number profile for a simulation of 50 single copy number changes with Gaussian noise, a tetraploid reference genome, and allowing for copy number changes of up to two per event. Bottom: clonal similarity as a function of scaling. The inferred scaling factor is unique and correctly has a value of approximately 4, reflecting the tetraploid reference genome.

We see that in every case, we are able to correctly recover our absolute copy number profile from the relative copy number dataset, suggesting that this methodology is applicable to a range of different scenarios.

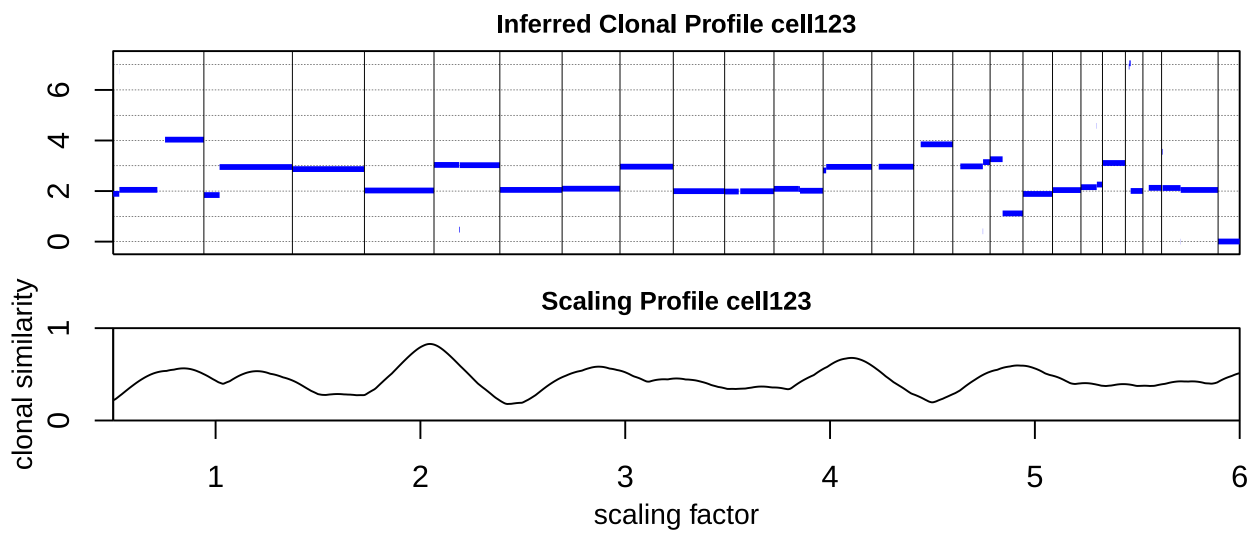
9 Analysing Real Data

Having seen that optimizing our proposed clonal similarity metric does indeed recover our original absolute copy numbers, we use a similar strategy to generate absolute copy number profiles for the experimental datasets provided.
As the data is given in the format [chromosome, start, end, cn], we first convert it to a dataframe of [chromosome, length, cn] for compatibility with previously described functions. This is achieved with the function parse_segs(), which also fills in regions for which we do not have copy number information with NA. As this function is not part of the main report it is given in the appendix together with the function length_to_sec() which performs the opposite data conversion.

```
segs = readRDS('relative_segment_tables.rds')
news = parse_segs(segs)
```

For each of the five samples provided, we can now run the test_proximity() function.

```
rds_1 = test_proximity(news[[1]], main = names(news)[1])
```

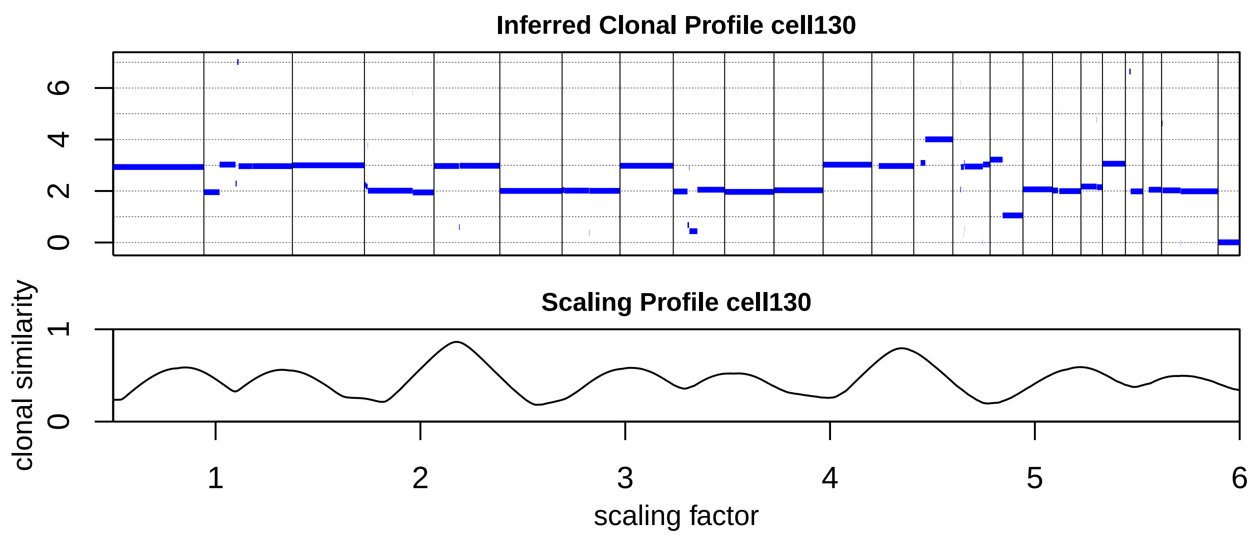


```
## scaling: 2.045 clonal similarity: 0.8288114
```

Figure 8: Inferred absolute copy number profile (top) and scaling profile (bottom) for experimental sample 1.

We see that for cell123 using a scaling factor of very close to 2, we obtain a high clonal similarity of 0.83 and a genome that appears to be diploid but with significant genome duplications. These consist mostly of complete duplications of individual chromosomes.

```
rds_2 = test_proximity(news[[2]], main = names(news)[2])
```

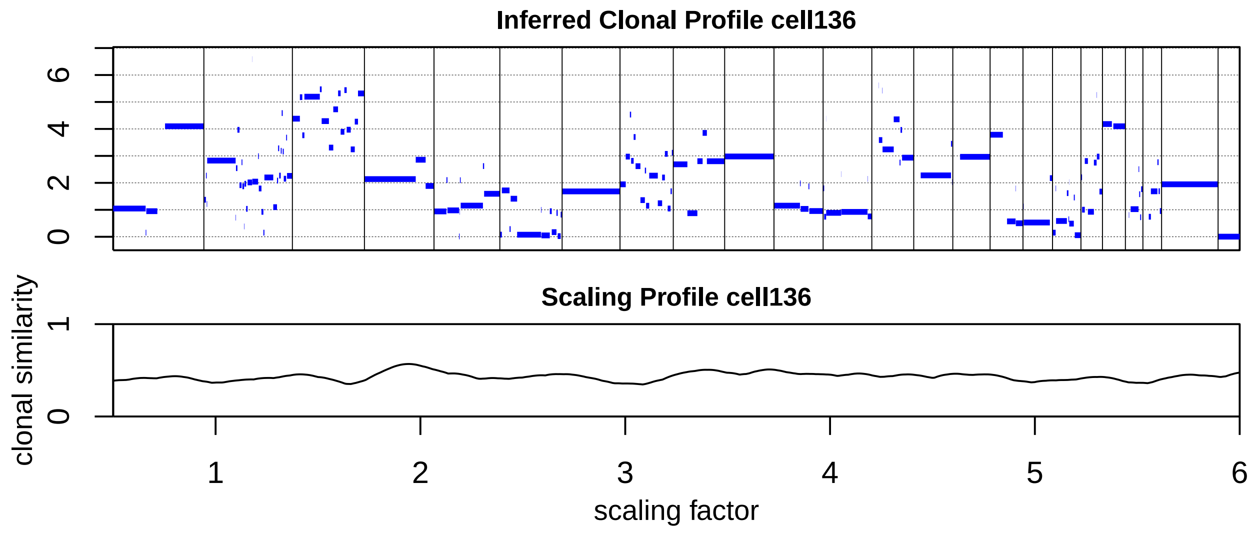


```
## scaling: 2.175 clonal similarity: 0.8633433
```

Figure 9: Inferred absolute copy number profile (top) and scaling profile (bottom) for experimental sample 2.

cell130 has a clear peak at a scaling factor of 2.175, giving again a high clonal similarity of 0.86 and what appears to be either a diploid genome with significant genome duplications or a triploid genome with large deletions.

```
rds_3 = test_proximity(news[[3]], main = names(news)[3])
```

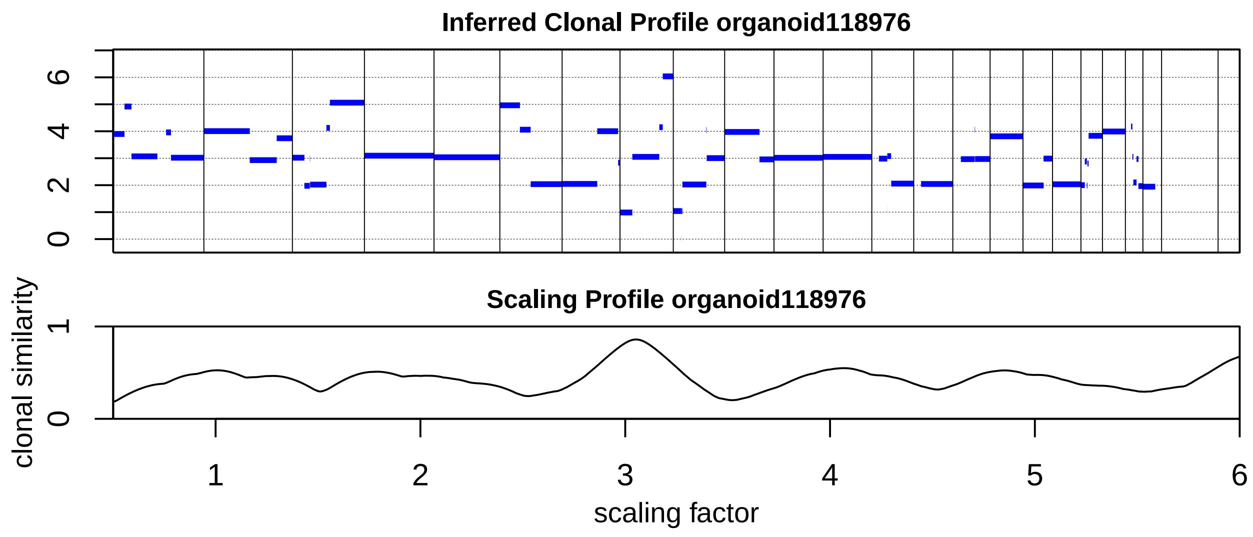


```
## scaling: 1.945 clonal similarity: 0.5684777
```

Figure 10: Inferred absolute copy number profile (top) and scaling profile (bottom) for experimental sample 3.

cell136 does not have a very clear clonal similarity peak, and the optimum scaling factor of 1.945 only gives a clonal similarity of 0.568. We are thus not particularly confident in our estimated absolute copy number profile for this sample. Looking at the inferred copy number profile we observe that it is very fragmented with fragments at an almost continous range of copy numbers, explaining our difficulty in accurately assigning absolute copy numbers. However, we note that the major fragments seem to be separated in copy number by approximately integer values, lending support to our proposed scaling of approximately 2.

```
rds_4 = test_proximity(news[[4]], main = names(news)[4])
```

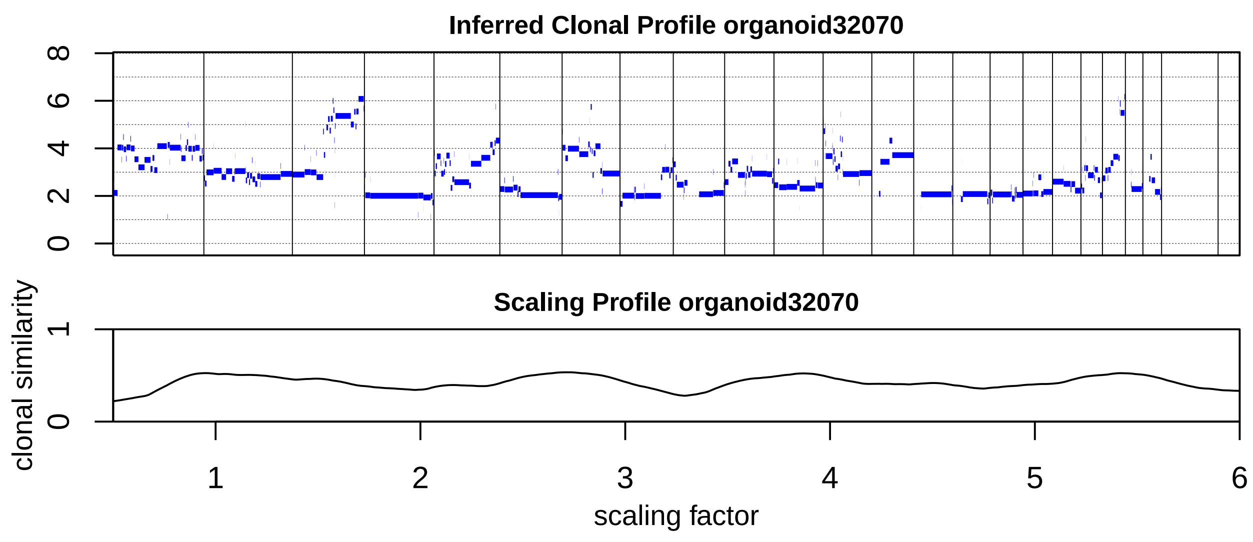


```
## scaling: 3.055 clonal similarity: 0.8590133
```

Figure 11: Inferred absolute copy number profile (top) and scaling profile (bottom) for experimental sample 4.

For organoid118976 we observe larger copy number fragments and get a very clear peak at a scaling factor of 3.055 with a high clonal similarity of 0.86. This suggests that we could be looking at a triploid reference genome with various copy number gains and losses.

```r
rds_5 = test_proximity(news[[5]], main = names(news)[5])
```



```
## scaling: 2.72 clonal similarity: 0.5343449
```

Figure 12: Inferred absolute copy number profile (top) and scaling profile (bottom) for experimental sample 5.

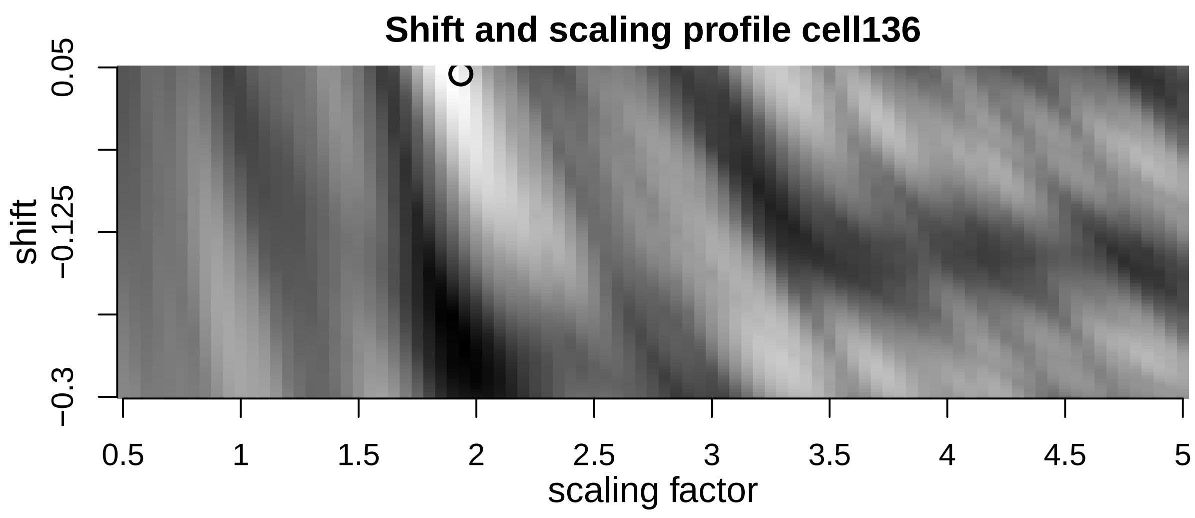
For organoid32070 there is very little difference between scaling factors of 1.8, 2.7, and 3.8 which all give rise to relatively modest clonal similarities of approximately 0.53. We are thus again not particularly confident in our absolute copy number estimates.

To further investigate cell136 and organoid32070 where we do not appear to get a confident estimate of absolute copy numbers, we use a slightly different apporach where we also allow for an additive shift in the signal (figures 13 & 14). This corresponds to accounting for additive noise, based on the assumption that our experimental signal has the approximate form $R = \alpha S + \beta$. Assuming β to include baseline noise, we expect to have to subtract noise from our relative signal to obtain the true signal and therefore let the shift parameter run from -0.3 to +0.05. This corresponds to allowing for β values from -0.05 to 0.3. This is achieved with the function test_proximity_ss() which is exceedingly similar to test_proximity() in its structure, and it is therefore included in the appendix rather than the main text.

We see that allowing for a shift in copy number prior to scaling does allow us to improve our clonal similarities, suggesting a better fit to a clonal profile. We also note that the scaling parameters change only modestly from to 1.945 to 1.9 for sample 3 and from 2.72 to 3.15 for sample 5. That is, the scalings remain near the same integer values with and without additive noise, and the two methods thus suggest similar ploidies for the reference genomes.

In summary we find that by optimizing a Euclidean clonal similarity metric, we can recover absolute copy number profiles from simulated relative copy number profiles. This is true for both diploid and multiploid reference genomes, a range of different copy number changes, and when including Gaussian noise. When applying this methodology to experimental data, we similarly recover clonal copy number profiles with a high clonal similarity for a number of samples. Two samples have low maximum clonal similarities with no clear peak, suggesting that the present method does not give a reliable absolute copy number profile for these samples. However, this can be partially remedied by allowing for a shift in the copy number data, accounting for e.g. a systematic baseline signal.

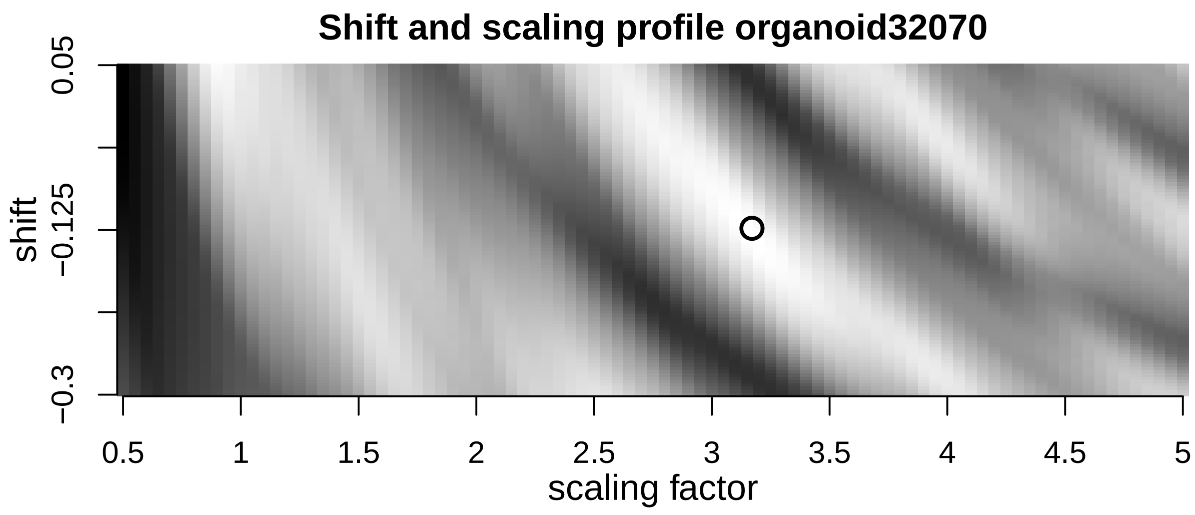
```r
rds_3 = test_proximity_ss(news[[3]], main = names(news)[3])
```



```
## shift: 0.043 scale: 1.9 clonal similarity: 0.5831508
```

Figure 13: Heatmap illustrating the dependence of clonal similarity on the shift and scaling used to generate an absolute copy number profile for experimental sample 3. Black circle indicates point of maximum clonal similarity.

```r
rds_5 = test_proximity_ss(news[[5]], main = names(news)[5])
```



```
## shift: -0.125 scale: 3.15 clonal similarity: 0.5487136
```

Figure 14: Heatmap illustrating the dependence of clonal similarity on the shift and scaling used to generate an absolute copy number profile for experimental sample 5. Black circle indicates point of maximum clonal similarity.

Appendix

```r
parse_segs = function(segs){
  #given a list of dataframes with structure [chr, start, end, cn]
  #returns a list of dataframes with structure [chr, length, cn]
  #fills in segments for which we don't have data with NA

  newsegs = list()
  chr_sizes = read.table(chr_size_file)
  for (i in 1:length(segs)){
    seg = segs[[i]]
    seg[,'start'] = as.numeric(seg[,'start'])
    seg[,'end'] = as.numeric(seg[,'end'])
    seg[,'cn'] = as.numeric(seg[,'cn'])

    for (j in c(1:22,'X','Y')){ #standardize chromosome labels
      seg[seg[,'chr'] == j, 'chr'] = paste0('chr',j)
    }
    newseg = data.frame('chr' = c(seg[1,'chr']),
                        'length' = c(seg[1,'end'] - seg[1,'start']+1),
                        'cn' = c(seg[1,'cn']), stringsAsFactors = FALSE)
    if (seg[1,'start'] != 1){ #if we don't have data for beginning of chr1, add NA
      newseg = rbind(list(seg[1,'chr'], seg[1,'start']-1,  NA), newseg)
    }
    if ((seg[1,'end']+1 != seg[2,'start']) & (seg[1, 'chr'] == seg[2, 'chr'])){
      #if we have a gap between segments, add NA
      newseg = rbind(newseg, list(seg[1,'chr'],
                              seg[2,'start'] - seg[1,'end']-1,  NA))
    }
    for (n in 2:(dim(seg)[1]-1)){ #run through segments
      if (seg[n, 'chr'] != seg[n-1, 'chr']){
        #if new chromsome
        if (seg[n,'start'] != 1){ #if no data for beginning of chr, add NA
          newseg = rbind(newseg, list(seg[n,'chr'], seg[n,'start']-1,  NA))
        }
      }
      newseg = rbind(newseg,
                     list(seg[n,'chr'], seg[n,'end'] - seg[n,'start']+1,
                     seg[n,'cn']))
      if ((seg[n,'end']+1 != seg[n+1,'start']) &
          (seg[n, 'chr'] == seg[n+1, 'chr'])){
        #if we have a gap between segments, add NA
        newseg = rbind(newseg, list(seg[n,'chr'],
                       seg[n+1,'start'] - seg[n,'end']-1,  NA))
      }
      if (seg[n, 'chr'] != seg[n+1, 'chr']){
        #if we don't have data for end of chr, add NA
        if (seg[n,'end'] != chr_sizes[chr_sizes[,1] == seg[n,'chr'],2]){
          newseg = rbind(newseg,list(seg[n,'chr'],
                    chr_sizes[chr_sizes[,1] == seg[n,'chr'],2]-seg[n,'end'],NA))
        }
      }
    }
    #fill out the rest of the final chromosome
    newseg = rbind(newseg, list(seg[dim(seg)[1],'chr'],
                                seg[dim(seg)[1],'end']-seg[dim(seg)[1],'start']+1,
                                seg[dim(seg)[1],'cn']))
    if (seg[dim(seg)[1],'end'] != chr_sizes[chr_sizes[,1] == seg[n,'chr'],2]){
      newseg = rbind(newseg, list(seg[dim(seg)[1],'chr'],
                                chr_sizes[chr_sizes[,1]==seg[dim(seg)[1],'chr'],2]
                                -seg[dim(seg)[1],'end'], NA))
    }
    newsegs[[i]] = newseg #add to list of segments with new format
  }
  names(newsegs) = names(segs)
  return(newsegs)
}

length_to_seg = function(results){
  #given a dataframe with structure [chr, length, cn]
  #returns a dataframe with structure [chr, start, end, cn]
  segs = data.frame('chr' = c(results[1,'chr']), 'start' = c(1),
                    'end' = c(results[1,'length']), 'cn' = c(results[1,'cn']),
                    stringsAsFactors = 0)
  x = results[1,'length'] #use x as positional counter along chromsome

  for (i in 2:dim(results)[1]){
    if (results[i, 'chr'] == results[i-1, 'chr']){ #if same chromosome
      segs[i,] = list(results[i,'chr'],x+1,x+results[i,'length'],results[i,'cn'])
      x = x + results[i, 'length']
    }else{ #if starting new chromsome, reset counter
      segs[i,] = list(results[i,'chr'], 1, results[i,'length'], results[i,'cn'])
      x = results[i, 'length']
    }
  }
  return(segs)
}

test_proximity_ss = function( results, scale_factors = 'default',
                              shift_factors = 'default', main='default' ){
  #finds absolute copy numbers using both scaling and shifting
  res = results[ !is.na(results[,'cn']), ] #consider only segments with data
  if (scale_factors == 'default'){
    scale_factors = seq( 0.5, 5, l=91 ) #go from .5ploid to pentaploid reference
  }
  if (shift_factors == 'default'){
    shift_factors = seq( -0.3,0.05, l=101) #assume noise is additive
  }
  sims = matrix(,length(shift_factors), length(scale_factors))
  test_res=res
  for (shift in 1:length(shift_factors)){
    for (scale in 1:length(scale_factors)){
      #consider combinations of scalings and shifts
      test_res[,'cn'] = (res[,'cn']+shift_factors[shift])*scale_factors[scale]
      sim = clonal_proximity(test_res) #quantify clonal similarity
      sims[shift,scale] = sim #store in matrix
    }
  }

  simmax = max(sims)
  param_max = which(sims==simmax, arr.ind=1) #find optimum shift & scaling
  shiftmax = shift_factors[param_max[1,1]]
  scalemax = scale_factors[param_max[1,2]]

  par( mfrow = c(1,1), mar = c(4,4,2,2))
  #plot heatmap of clonal similarities
  image(t(sims), xlab='', ylab='', axes=0, col=grey(seq(0, 1, length = 256)),
        main = paste0('Shift and scaling profile ', main) )
  axis(1, at = seq(0, 1, length = 10),
       labels=seq(scale_factors[1], scale_factors[length(scale_factors)], l=10)
       ,srt=45,tick=TRUE)
  axis(2, at = seq(0, 1, length = 5),
       labels=seq( shift_factors[1],shift_factors[length(shift_factors)], l=5)
       ,srt=45,tick=TRUE)
  title(ylab="shift", line=2, cex.lab=1.2)
  title(xlab="scaling factor", line=2, cex.lab=1.2)
  draw.circle(param_max[1,2]/length(scale_factors),
              param_max[1,1]/length(shift_factors),
              0.01, border='black', lwd=2)

  #write a summary
  cat('shift:', shiftmax, 'scale:', scalemax,'clonal similarity:', simmax)

  return(list('results'=results, 'scaling'=scalemax,
              'shift'=shiftmax, 'clonal'=simmax))
}
```