Copy Number Pipeline: ASCAT and CINdex

Juan Lorell

2024-12-14

Background of Data

The data for this analysis contains data GSE87048. The data contains CEL file from the micro array platform Affymetrix SNP 6.0 Human totaling 100 raw data that was processed using steps shown later . The specific distribution of the data can be seen directly in the report (please contact author). The end goal of the project, specifically for the R portion of the coding was to create an input suitable for machine learning prediction and also to create a pipeline analysis for copy number research.

ASCAT

ASCAT is a package developed by van Loo et al. (2010). which produces copy number analysis from either SNP array data, WGS, or WES. Although old, the package is still used to this day as a good analysis software to determine the copy number, ploidy, and also purity from a given sample. However, the input for this software is the LRR and BAF file from each sample which cannot be processed by the software. To get the input, the authors of the software have given instructions on how to process the raw .CEL data files.

Package Information

Below was the package that are used and also dependencies to the packages used. This section of the code will give the code for installing the packages and calling the library. Please delete the "#" if the package is not downloaded yet into the R environment. Please visit https://github.com/VanLoo-lab/ascat for more information.

```r
setwd("~/brca/multi/COPY_NUMBER_ANALYSIS")
#BiocManager::install("GenomicRanges")
#BiocManager::install("IRanges")
#devtools::install_github('VanLoo-lab/ascat/ASCAT')
library(ASCAT)
library(GenomicRanges)
```

```
## Loading required package: stats4

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, saveRDS, setdiff,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:utils':
##
##     findMatches
```

```
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
```

```
## Loading required package: IRanges
```

```
## Loading required package: GenomeInfoDb
```

```r
library(IRanges)
```

Pre-ASCAT

The author of ASCAT suggested that the requisition of the LRR and BAF data for each is done using the PennCNV-Affy pipeline which is first processed by the Affymetrix Power Tool (AFP). Do note that the use of both tools can only be done in a Linux terminal with the choice of using Linux Sub-system for Windows (LSW) being possible to run AFP, however the use of PennCNV has not been tested for LSW. Attached below are the code for running both PennCNV and AFP that was used. Remove and copy-paste the code in a linux (or LSW) command terminal to get the input for ASCAT. Please take into account that additional reference data is needed to allow mapping of the CEL data file by reading the tutorial in the link above.

```r
#Step 1: genotyping
#put/directory/here/apt-probeset-genotype -c put/directory/here/GenomeWideSNP_6.cdf -a birdseed --read-


#Step2: summarizing
#put/directory/here/apt-probeset-summarize --cdf-file put/directory/here/GenomeWideSNP_6.cdf --analysis


#Step 3: get LRR and BAF
#put/directory/here/penncnv/LibFiles/penncnv/gw6/bin/normalize_affy_geno_cluster.pl put/directory/here/
```

The result of step 3 can then be directly used for ASCAT, there is no need for normalization as it is done in the second step of the section above. Later on, ASCAT will also normalize the results using its own algorithm which will be shown in the next part.

ASCAT Data Preparation

Data prep was done by first reading the data from the subsection above into r. The input for ASCAT, although it is LRR and BAF, must be in its own separate CSV (data frame if you don't want to write the data). As such, the code below first takes the chromosome location and specific nucleotide base where the DNA will bind to the specific SNP tags.This will then be used as a base for creating an empty data frame that could then be used to call data from LRR and BAF which are then matched using the data from SNPpos. The call of each sample is done using a for loop which is then concatenated together into the pre-made data frame. In LRR specifically, there is a normalization step using the mean of each SNP position. After that, the data is then made into txt, which is optional.

```r
#read data and define data frame
lrrbaf = read.table("lrr_baf.txt", header = T, sep = "\t", row.names=1)

SNPpos <- lrrbaf[, c(1, 2)]

# Log R preparation
logR_normal <- data.frame(row.names = rownames(SNPpos))
logR_tumor <- data.frame(row.names = rownames(SNPpos))

#separate tumor and normal
for (i in seq(3, ncol(lrrbaf), by = 2)) {
  s_logr <- sub(".CEL.Log.R.Ratio", "", colnames(lrrbaf)[i])

  # Determine if the sample is tumor or normal
  if (grepl("_C\\d+", s_logr)) {
    # Normal sample
    Normal_LogR = lrrbaf[rownames(SNPpos), i, drop=F]
    colnames(Normal_LogR) = s_logr

    CNprobes = substring(rownames(SNPpos), 1, 2) == "CN"

    Normal_LogR[CNprobes, 1] = Normal_LogR[CNprobes, 1] - mean(Normal_LogR[CNprobes, 1], na.rm=T)
    Normal_LogR[!CNprobes, 1] = Normal_LogR[!CNprobes, 1] - mean(Normal_LogR[!CNprobes, 1], na.rm=T)
    Normal_LogR = round(Normal_LogR, 4)

    logR_normal <- cbind(logR_normal, Normal_LogR)

  } else if (grepl("_T\\d+", s_logr)) {
    # Tumor sample
    Tumor_LogR = lrrbaf[rownames(SNPpos), i, drop=F]
    colnames(Tumor_LogR) = s_logr

    CNprobes = substring(rownames(SNPpos), 1, 2) == "CN"

    Tumor_LogR[CNprobes, 1] = Tumor_LogR[CNprobes, 1] - mean(Tumor_LogR[CNprobes, 1], na.rm=T)
    Tumor_LogR[!CNprobes, 1] = Tumor_LogR[!CNprobes, 1] - mean(Tumor_LogR[!CNprobes, 1], na.rm=T)
    Tumor_LogR = round(Tumor_LogR, 4)

    logR_tumor <- cbind(logR_tumor, Tumor_LogR)
  }
}

# BAF preparation
baf_normal <- data.frame(row.names = rownames(SNPpos))
baf_tumor <- data.frame(row.names = rownames(SNPpos))

#separate tumor and normal
for (i in seq(4, ncol(lrrbaf), by = 2)) {
  s_baf <- sub(".CEL.B.Allele.Freq", "", colnames(lrrbaf)[i])

  # Determine if the sample is tumor or normal
  if (grepl("_C\\d+", s_baf)) {
    # Normal sample
    Normal_BAF = lrrbaf[rownames(SNPpos), i, drop=F]
    colnames(Normal_BAF) = s_baf

    Normal_BAF[Normal_BAF == 2] = NA

    baf_normal <- cbind(baf_normal, Normal_BAF)

  } else if (grepl("_T\\d+", s_baf)) {
    # Tumor sample
    Tumor_BAF = lrrbaf[rownames(SNPpos), i, drop=F]
    colnames(Tumor_BAF) = s_baf

    Tumor_BAF[Tumor_BAF == 2] = NA

    baf_tumor <- cbind(baf_tumor, Tumor_BAF)
  }
}
#write the result

write.table(cbind(SNPpos,baf_tumor),paste("tumor.BAF.txt"),sep="\t",row.names=T,col.names=NA,quote=F)
write.table(cbind(SNPpos,logR_tumor),paste("tumor.LogR.txt"),sep="\t",row.names=T,col.names=NA,quote=F)
write.table(cbind(SNPpos,baf_normal),paste("normal.BAF.txt"),sep="\t",row.names=T,col.names=NA,quote=F)
write.table(cbind(SNPpos,logR_normal),paste("normal.LogR.txt"),sep="\t",row.names=T,col.names=NA,quote=
```

If the sample only has tumor data, the code below can be used.

```
#log_r prep
logR <- data.frame(row.names = rownames(SNPpos))
for (i in seq(3, ncol(lrrbaf), by = 2)) {
  s_logr <- sub(".CEL.Log.R.Ratio","",colnames(lrrbaf)[i])
  Tumor_LogR = lrrbaf[rownames(SNPpos),i,drop=F]
  colnames(Tumor_LogR) = s_logr

  CNprobes = substring(rownames(SNPpos),1,2)=="CN"

  Tumor_LogR[CNprobes,1] = Tumor_LogR[CNprobes,1]-mean(Tumor_LogR[CNprobes,1],na.rm=T)
  Tumor_LogR[!CNprobes,1] = Tumor_LogR[!CNprobes,1]-mean(Tumor_LogR[!CNprobes,1],na.rm=T)
  Tumor_LogR = round(Tumor_LogR,4)

  logR <- cbind(logR, Tumor_LogR)
}

#BAF prep
baf <- data.frame(row.names = rownames(SNPpos))
for (i in seq(4, ncol(lrrbaf), by = 2)) {
  s_baf <- sub(".CEL.B.Allele.Freq","",colnames(lrrbaf)[i])
  Tumor_BAF = lrrbaf[rownames(SNPpos),i,drop=F]
  colnames(Tumor_BAF) = s_baf

  Tumor_BAF[Tumor_BAF==2]=NA

  baf<-cbind(baf, Tumor_BAF)
```

```r
}
#write the result
write.table(cbind(SNPpos,baf),paste("tumor.BAF.txt"),sep="\t",row.names=T,col.names=NA,quote=F)
write.table(cbind(SNPpos,logR),paste("tumor.LogR.txt"),sep="\t",row.names=T,col.names=NA,quote=F)
```

ASCAT run

Data Calling

Data calling is done by first importing the birdseed algorithm report from the pre-processing step. This will be used to get the gender of each patient in case those are wanted, which in this case is. The command used will separate the gender and transform the categorization into the allosomal form. Then, we can read the data from the previous step. Do note that we can call the data frame directly from the r environment. The calling using txt was done to easily see the data in excel.

```r
gender <- read.table("./geno/birdseed.report.txt", sep="\t", skip=66, header=T)
sex <- as.vector(gender[,"computed_gender"])
sex[sex == "female"] <- "XX"
sex[sex == "male"] <- "XY"
sex[sex == "unknown"] <- "XX"

# Load data into ASCAT
ascat.bc <- ascat.loadData(Tumor_LogR_file = "tumor.LogR.txt",
                            Tumor_BAF_file = "tumor.BAF.txt",
                            Germline_LogR_file = "normal.LogR.txt",
                            Germline_BAF_file = "normal.BAF.txt",
                            gender = sex,
                            genomeVersion = "hg19")
```

```
## [1] Reading Tumor LogR data...
## [1] Reading Tumor BAF data...
## [1] Reading Germline LogR data...
## [1] Reading Germline BAF data...
## [1] Registering SNP locations...
## [1] Splitting genome in distinct chunks...
```

Normalization

Normalization is done by ASCAT for the LRR data specifically for SNP array. This is done by comparing it with the normalized GC content and replication timing from the SNP array platform. As the platform used, as mentioned above, was SNP array 6.0 human, the authors has already made the needed file. The authors has also made some correction files for a few other SNP array platform which is convenient as the way to generate them is computationally expensive.

```r
ascat.nor = ascat.correctLogR(ascat.bc,
                             GCcontentfile = "GC_nor.txt",
                             replictimingfile = "rt_nor.txt")
```

Predicting Germline data

As the sample has germline data for comparison this code was not run. However, in the case a normal sample is not collected alongside tumor sample, then this code can be used to predict the state/signal of the tumor may be. Do keep in mind that this might not result in accurate or reliable data.

```r
##gg = ascat.predictGermlineGenotypes(ascat.nor,
##                                    platform = "AffySNP6",
##                                    img.dir="./res/germline",
##                                    img.prefix = "germline_")
```

Segment Data

```
ascat.seg = ascat.aspcf(ascat.nor,
                        ascat.gg=NULL,
                        out.dir="./res")
```

Run main ASCAT program

```
ascat.output = ascat.runAscat(ascat.seg,
                              img.dir="./res/combi",
                               write_segments = T)
```

Visualization of ASCAT

Each step of ASCAT can be visualized using the code below. However, it is suggested to dp this last or as a background job while running CINdex as it is time consuming.

```
#print raw data
#ascat.plotRawData(ascat.bc, img.dir = "./picture/before",img.prefix = "b_")

#print normalized data
#ascat.plotRawData(ascat.nor, img.dir = "./picture/normalized",img.prefix = "n_")

#print predicted germline
#ascat.plotRawData(gg, img.dir = "./picture/predicted",img.prefix = "p_")

#print segemented data
#ascat.plotSegmentedData(ascat.seg, img.dir="./picture/segmented", img.prefix = "s_")
```

Chromosome Instability Index (CINdex)

CINdex is a package developed by Song L et al.(2017), which process high throughput screening technology for copy number data analysis.The package itself has been in the BioC repository since 2016. The main idea of the package is to allow users to see the genomic instability of the sample for either overall condition or in a specific section of a chromosome.

Package information

The packages below are the packages listed as dependencies of CINdex. Please delete the "#" symbol if the package is not downloaded yet into the R environment For more information, please see the CINdex manual and attached files for more info in the BioConductor link: https://www.bioconductor.org/packages/release/bioc/html/CINdex.html.

```
#BiocManager::install("pd.genomewidesnp.6")
#BiocManager::install("rtracklayer")
#BiocManager::install("biovizBase")
#install.packages("R.utils")
#BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
#BiocManager::install("Homo.sapiens")
```

```r
#BiocManager::install("CINdex")
library(pd.genomewidesnp.6)
```

```
## Loading required package: Biostrings

## Loading required package: XVector

##
## Attaching package: 'Biostrings'

## The following object is masked from 'package:base':
##
##     strsplit

## Loading required package: RSQLite

## Loading required package: oligoClasses

## Welcome to oligoClasses version 1.68.0

## Loading required package: oligo

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## ================================================================================

## Welcome to oligo version 1.70.0

## ================================================================================

## Loading required package: DBI
```

```r
library(rtracklayer)
library(biovizBase) #needed for stain information
library(R.utils)
```

```
## Loading required package: R.oo

## Loading required package: R.methodsS3

## R.methodsS3 v1.8.2 (2022-06-13 22:00:14 UTC) successfully loaded. See ?R.methodsS3 for help.

## R.oo v1.27.0 (2024-11-01 18:00:02 UTC) successfully loaded. See ?R.oo for help.

##
## Attaching package: 'R.oo'

## The following object is masked from 'package:R.methodsS3':
##
##     throw

## The following object is masked from 'package:GenomicRanges':
##
##     trim

## The following object is masked from 'package:IRanges':
##
##     trim
## The following objects are masked from 'package:methods':
##
##     getClasses, getMethods

## The following objects are masked from 'package:base':
##
##     attach, detach, load, save

## R.utils v2.12.3 (2023-11-18 01:00:02 UTC) successfully loaded. See ?R.utils for help.

##
## Attaching package: 'R.utils'

## The following object is masked from 'package:oligoClasses':
##
##     isPackageLoaded

## The following object is masked from 'package:utils':
##
##     timestamp

## The following objects are masked from 'package:base':
##
##     cat, commandArgs, getOption, isOpen, nullfile, parse, use, warnings
```

```r
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

```
## Loading required package: GenomicFeatures

## Loading required package: AnnotationDbi
```

```r
library(Homo.sapiens)
```

```
## Loading required package: OrganismDbi

## Loading required package: GO.db

##

## Loading required package: org.Hs.eg.db

##
```

```r
library(org.Hs.eg.db)
library(AnnotationDbi)
library(GenomicRanges)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:OrganismDbi':
##
##     select

## The following object is masked from 'package:AnnotationDbi':
##
##     select

## The following object is masked from 'package:oligo':
##
##     summarize
## The following object is masked from 'package:Biobase':
##
##     combine

## The following objects are masked from 'package:Biostrings':
##
##     collapse, intersect, setdiff, setequal, union

## The following object is masked from 'package:XVector':
##
##     slice

## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union

## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect

## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(CINdex)

#note: some package may cause conflict, be carefull when running this code block.
```

Data Preparation

This part is separated into 5 parts which is used to call and prepare the data so it can be used as a suitable input for CINdex

Prepare probe annotation file

In this part, the CINdex manual gives several options. However, the first option will be chosen as it is more through in the steps to filter and select the annotation file.

```r
#connect to the underlying SQLite database that is part of the pd.genomewidesnp.6 package
con <- db(pd.genomewidesnp.6)
# get the copy number probes
cnv <- dbGetQuery(con, "select man_fsetid, chrom, chrom_start, chrom_stop,
strand from featureSetCNV;")
head(cnv, n =3) #print first few rows
```

```
##   man_fsetid chrom chrom_start chrom_stop strand
## 1  CN_477984     1       62140      62165      0
## 2  CN_473963     1       61723      61748      0
## 3  CN_473964     1       61796      61821      0
```

```
#get the SNP probes
snp <- dbGetQuery(con, "select man_fsetid, dbsnp_rs_id, chrom, physical_pos,
strand from featureSet;")
head(snp, n=3)
```

```
##      man_fsetid dbsnp_rs_id chrom physical_pos strand
## 1 SNP_A-2131660   rs2887286     1      1156131      0
## 2 SNP_A-1967418   rs1496555     1      2234251      0
## 3 SNP_A-1969580  rs41477744     1      2329564      0
```

After Downloading all the reference data we can then transform them into a GenomicRange data. This was done using the code below.

```
#function to convert Copy number data into GRanges object
convert.to.gr.cnv <- function(cnv2) {
  cnv2$chrom <- paste0("chr", cnv2$chrom)
  # subset out SNPs with missing location
  cnv2 <- cnv2[!(is.na(cnv2$chrom) | is.na(cnv2$chrom_start)),]
  # convert strand info to +,-,*
  cnv2$strand[is.na(cnv2$strand)] <- 2
  cnv2$strand <- cnv2$strand + 1
  cnv2$strand <- c("+", "-", "*")[cnv2$strand]
  #convert into GRanges object
  cnv2.gr <- GRanges(cnv2$chrom, IRanges(cnv2$chrom_start,cnv2$chrom_stop),
  cnv2$strand, ID = cnv2$man_fsetid)
  return(cnv2.gr)
}

#function to convert SNP data into GRanges object
convert.to.gr.snp <- function(snp2) {
  # make chromosomes the same as a chain file
  snp2$chrom <- paste0("chr", snp2$chrom)
  # subset out SNPs with missing location
  snp2 <- snp2[!(is.na(snp2$chrom) | is.na(snp2$physical_pos)),]
  # convert strand info to +,-,*
  snp2$strand[is.na(snp2$strand)] <- 2
  snp2$strand <- snp2$strand + 1
  snp2$strand <- c("+", "-", "*")[snp2$strand]
  snp2.gr <- GRanges(snp2$chrom, IRanges(snp2$physical_pos,snp2$physical_pos),
  snp2$strand, ID = snp2$man_fsetid,
  dbsnp = snp2$dbsnp_rs_id)
  return(snp2.gr)
}

# convert this copy number data from into a GRanges object
cnv.gr <- convert.to.gr.cnv(cnv2 = cnv)
head(cnv.gr, n=3)
```

```
## GRanges object with 3 ranges and 1 metadata column:
##       seqnames      ranges strand |          ID
##          <Rle>   <IRanges>  <Rle> | <character>
##    [1]     chr1 62140-62165      + |   CN_477984
##    [2]     chr1 61723-61748      + |   CN_473963
##    [3]     chr1 61796-61821      + |   CN_473964
##   -------
##   seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

```
# convert this SNP data from into a GRanges object
snp.gr <- convert.to.gr.snp(snp2 = snp)
head(snp.gr, n=3)
```

```
## GRanges object with 3 ranges and 2 metadata columns:
##       seqnames    ranges strand |          ID       dbsnp
##          <Rle> <IRanges>  <Rle> | <character> <character>
##    [1]     chr1   1156131      + | SNP_A-2131660   rs2887286
##    [2]     chr1   2234251      + | SNP_A-1967418   rs1496555
##    [3]     chr1   2329564      + | SNP_A-1969580  rs41477744
##   -------
##   seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

According to the current version of CINdex (version 1.32.0), the current package does not support the analysis of allosome and mitochondrial chromosome. As such, it is necessary to remove them from the dataframe which can be done using the code below if the sample contains said data.

```
#subset only those probes that are in autosomes
snpgr.19.auto <- subset(snp.gr, seqnames(snp.gr) %in% c("chr1",
                                                        "chr2", "chr3","chr4",
                                                        "chr5", "chr6", "chr7",
                                                        "chr8", "chr9", "chr10",
                                                        "chr11", "chr12", "chr13",
                                                        "chr14", "chr15", "chr16",
                                                        "chr17", "chr18","chr19",
                                                        "chr20", "chr21", "chr22"))

#subset only those probes that are in autosomes
cnvgr.19.auto <- subset(cnv.gr, seqnames(cnv.gr) %in% c("chr1",
                                                        "chr2", "chr3","chr4",
                                                        "chr5", "chr6", "chr7",
                                                        "chr8", "chr9", "chr10",
                                                        "chr11", "chr12", "chr13",
                                                        "chr14", "chr15", "chr16",
                                                        "chr17", "chr18","chr19",
                                                        "chr20", "chr21", "chr22"))
```

Reference Genome Data Calling

The refernce data is done by importing the cytoband and staining information from the hg19 reference genome. This was done using the rtrack package, thgough there are a few other options that can be used such as direct importing from USCS genome browser.

```
# create a query against a UCSC Table browser
query <- rtracklayer::ucscTableQuery("hg19", table = "cytoBand")
table1 <- rtracklayer::getTable(query) # retrieve table
head(table1)
```

```
##   chrom chromStart chromEnd   name gieStain
## 1  chr1          0  2300000 p36.33     gneg
## 2  chr1    2300000  5400000 p36.32   gpos25
## 3  chr1    5400000  7200000 p36.31     gneg
## 4  chr1    7200000  9200000 p36.23   gpos25
## 5  chr1    9200000 12700000 p36.22     gneg
## 6  chr1   12700000 16200000 p36.21   gpos50
```

```
#Add an extra column with strand information
table1$Strand <- c("*")
## Convert object into GRanges object
table1.gr <- GRanges(table1$chrom,
IRanges(table1$chromStart, table1$chromEnd),
table1$Strand,
table1$name, table1$gieStain)
head(table1.gr, n = 3)
```

```
## GRanges object with 3 ranges and 2 metadata columns:
##       seqnames          ranges strand | table1$name table1$gieStain
##          <Rle>       <IRanges>  <Rle> | <character>     <character>
##   [1]     chr1       0-2300000      * |      p36.33            gneg
##   [2]     chr1 2300000-5400000      * |      p36.32          gpos25
##   [3]     chr1 5400000-7200000      * |      p36.31            gneg
##   -------
##   seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

```
hg19.ucsctrack<-table1.gr
#Save this object for future use
#save(table1.gr, file = "hg19.ucsctrack.RData")
```

Clinical Data Calling

CINdex can be used to compare different patients data. As such, having samples that have differing conditions can be compared together. For CINdex, this is done through mostly box plot.

```
clin<-read.csv("./clinical.csv", header= TRUE)
str(clin)# check structure
```

```
## 'data.frame':    99 obs. of  2 variables:
##  $ Sample: chr  "GSM2319836_T1_Normal" "GSM2319837_T2" "GSM2319838_T4_LumA" "GSM2319839_T5_HER2" ..
##  $ Label : chr  "Normal" "Other" "LumA" "HER2" ...
```

```
class(clin)#check object type
```

```
## [1] "data.frame"
```

As the class is not what we want, we have to convert it. This is done using the code below

```
clin <- as.matrix(clin)
str(clin)
```

```
##  chr [1:99, 1:2] "GSM2319836_T1_Normal" "GSM2319837_T2" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "Sample" "Label"
```

```
class(clin)
```

```
## [1] "matrix" "array"
```

```r
print(head(clin))
```

```
##      Sample                   Label
## [1,] "GSM2319836_T1_Normal" "Normal"
## [2,] "GSM2319837_T2"        "Other"
## [3,] "GSM2319838_T4_LumA"   "LumA"
## [4,] "GSM2319839_T5_HER2"   "HER2"
## [5,] "GSM2319840_T6_LumB"   "LumB"
## [6,] "GSM2319841_T9"        "Other"
```

Calling Gene Annotation Data

Next, we need annotation data to translate the array id into Gene symbols. This is usually done with the TxDb database. Be mindful as there are a lot of annotation version, so make sure everything is the same.

```r
# Assign the UCSC hg19 TxDb object to Homo.sapiens
txdb<-TxDb(Homo.sapiens) <- TxDb.Hsapiens.UCSC.hg19.knownGene

z <- AnnotationDbi::select(txdb, keys(Homo.sapiens, "CDSID"),
            c("CDSID", "CDSCHROM", "CDSSTRAND", "CDSSTART", "CDSEND", "GENEID"), "CDSID")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```r
library(org.Hs.eg.db)

# Map GENEID to SYMBOL
gene_symbols <- AnnotationDbi::select(org.Hs.eg.db, z$GENEID, "SYMBOL", "ENTREZID")
```

```
## 'select()' returned many:1 mapping between keys and columns
```

```r
# Merge the result with your initial data
z <- merge(z, gene_symbols, by.x = "GENEID", by.y = "ENTREZID")


# Remove rows with missing values
z1 <- na.omit(object = z)
```

After that, the list of objects will be combined into a single matrix using the code below for easier calling.

```r
# extracting only the columns we want as a matrix
geneAnno <- cbind(z1$CDSCHROM, z1$CDSSTRAND, z1$CDSSTART, z1$CDSEND, z1$SYMBOL)
colnames(geneAnno) <- c("chrom","strand", "cdsStart", "cdsEnd", "GeneName")
#So this gene annotation file looks like this
head(geneAnno, n=3)
```

```
##      chrom   strand cdsStart   cdsEnd     GeneName
## [1,] "chr19" "-"    "58864294" "58864563" "A1BG"
## [2,] "chr19" "-"    "58864294" "58864563" "A1BG"
## [3,] "chr19" "-"    "58864294" "58864563" "A1BG"
```

```r
#Examining the class and structure of this oject
class(geneAnno)
```

```
## [1] "matrix" "array"
```

```r
str(geneAnno)
```

```
##  chr [1:6813214, 1:5] "chr19" "chr19" "chr19" "chr19" "chr19" "chr19" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "chrom" "strand" "cdsStart" "cdsEnd" ...
```

Transform segmented data

For CINdex, the data that are going to be used for the input needs to be a Grange object. As ASCAT does not store their data in Grange, the GenomicRanges package is used to convert the object type from list data type to Grange list. The conversion references the BioConductor document named "ASCAT to RaggedExperiment" by *King and Ramos (2024)* with a lot of modifications to the code.

```
raw<-ascat.output$segments_raw
print(head(raw))
```

```
##                  sample chr startpos   endpos nMajor nMinor    nAraw     nBraw
## 1 GSM2319836_T1_Normal   1    61735  3614480      2      1 2.087645 0.6812531
## 2 GSM2319836_T1_Normal   1  3614576  6784592      2      0 1.826204 0.0598151
## 3 GSM2319836_T1_Normal   1  6792896  7248814      2      1 1.343900 1.3439004
## 4 GSM2319836_T1_Normal   1  7251360 10502710      3      2 2.538515 2.5385150
## 5 GSM2319836_T1_Normal   1 10505132 15172626      2      1 2.003404 0.7124644
## 6 GSM2319836_T1_Normal   1 15172865 15785719      5      2 4.507723 1.7603901
```

```
raw<-dplyr::select(raw, -c("nMajor", "nMinor"))

class(raw)
```

```
## [1] "data.frame"
```

```
raw$value <- rowMeans(raw[, c("nAraw", "nBraw")])
head(raw)
```

```
##                  sample chr startpos   endpos    nAraw     nBraw     value
## 1 GSM2319836_T1_Normal   1    61735  3614480 2.087645 0.6812531 1.3844491
## 2 GSM2319836_T1_Normal   1  3614576  6784592 1.826204 0.0598151 0.9430094
## 3 GSM2319836_T1_Normal   1  6792896  7248814 1.343900 1.3439004 1.3439004
## 4 GSM2319836_T1_Normal   1  7251360 10502710 2.538515 2.5385150 2.5385150
## 5 GSM2319836_T1_Normal   1 10505132 15172626 2.003404 0.7124644 1.3579342
## 6 GSM2319836_T1_Normal   1 15172865 15785719 4.507723 1.7603901 3.1340568
```

```
raw_filtered <- raw[raw$chr %in% 1:22, ]

cin_data <- raw_filtered[, c("sample", "chr", "startpos", "endpos", "value")]
head(cin_data)
```

```
##                  sample chr startpos   endpos     value
## 1 GSM2319836_T1_Normal   1    61735  3614480 1.3844491
## 2 GSM2319836_T1_Normal   1  3614576  6784592 0.9430094
## 3 GSM2319836_T1_Normal   1  6792896  7248814 1.3439004
## 4 GSM2319836_T1_Normal   1  7251360 10502710 2.5385150
## 5 GSM2319836_T1_Normal   1 10505132 15172626 1.3579342
## 6 GSM2319836_T1_Normal   1 15172865 15785719 3.1340568
```

```
gr_list <- lapply(split(cin_data, cin_data$sample), function(df) {
  GRanges(seqnames = df$chr,
          ranges = IRanges(start = df$startpos, end = df$endpos),
          strand = "*",  # Assuming no strand information
          value = df$value)
})
input <- GRangesList(gr_list)
```

```r
class(input)
```

```
## [1] "CompressedGRangesList"
## attr(,"package")
## [1] "GenomicRanges"
```

```r
head(input)
```

```
## GRangesList object of length 6:
## $GSM2319836_T1_Normal
## GRanges object with 204 ranges and 1 metadata column:
##         seqnames            ranges strand |     value
##            <Rle>         <IRanges>  <Rle> | <numeric>
##     [1]        1     61735-3614480      * |  1.384449
##     [2]        1   3614576-6784592      * |  0.943009
##     [3]        1   6792896-7248814      * |  1.343900
##     [4]        1  7251360-10502710      * |  2.538515
##     [5]        1 10505132-15172626      * |  1.357934
##     ...      ...               ...    ... .       ...
##   [200]       21 17459013-33927768      * |   2.30992
##   [201]       21 33927886-48096957      * |   1.83294
##   [202]       22 16052528-25705314      * |   1.56287
##   [203]       22 25706918-25910756      * |   3.95678
##   [204]       22 25910771-51234455      * |   1.46329
##   -------
##   seqinfo: 22 sequences from an unspecified genome; no seqlengths
##
## ...
## <5 more elements>
```

To know which samples are used, the names of the samples in the input is printed using the code below. This code can also be used to select the data from which entry to which. Use another code to specifically select samples, but those are not given.

```r
names(input)
```

```
##  [1] "GSM2319836_T1_Normal"   "GSM2319837_T2"          "GSM2319838_T4_LumA"
##  [4] "GSM2319839_T5_HER2"     "GSM2319840_T6_LumB"     "GSM2319841_T9"
##  [7] "GSM2319842_T13_LumB"    "GSM2319843_T15_HER2"    "GSM2319844_T16_LumB"
## [10] "GSM2319845_T17_Normal"  "GSM2319846_T18"         "GSM2319847_T26_LumA"
## [13] "GSM2319848_T27_LumA"    "GSM2319849_T28_LumA"    "GSM2319850_T29_LumA"
## [16] "GSM2319851_T30_LumA"    "GSM2319852_T31_Normal"  "GSM2319853_T34_LumA"
## [19] "GSM2319854_T36_LumA"    "GSM2319855_T37_LumA"    "GSM2319856_T38_LumB"
## [22] "GSM2319857_T39"         "GSM2319858_T40_LumA"    "GSM2319859_T41_LumA"
## [25] "GSM2319860_T43_Normal"  "GSM2319861_T44_Normal"  "GSM2319862_T45_Basal"
## [28] "GSM2319863_T47_HER2"    "GSM2319864_T48_Normal"  "GSM2319865_T50_LumA"
## [31] "GSM2319866_T55_LumB"    "GSM2319867_T58_Normal"  "GSM2319868_T59LumA"
## [34] "GSM2319869_T62_LumA"    "GSM2319870_T69_LumA"    "GSM2319871_T71_LumA"
## [37] "GSM2319872_T73_LumB"    "GSM2319873_T74_LumA"    "GSM2319874_T75_HER2"
## [40] "GSM2319875_T76_HER2"    "GSM2319876_T77"         "GSM2319877_T79_LumB"
## [43] "GSM2319878_T80_Normal"  "GSM2319879_T81_LumA"    "GSM2319880_T82_LumB"
## [46] "GSM2319881_T83_LumB"    "GSM2319882_T85_HER2"    "GSM2319883_T94_LumB"
## [49] "GSM2319884_T95_HER2"    "GSM2319885_T97_Basal"   "GSM2319886_T98_LumA"
## [52] "GSM2319887_T105_LumB"   "GSM2319888_T106_LumA"   "GSM2319889_T108_HER2"
## [55] "GSM2319890_T110_LumA"   "GSM2319891_T113"        "GSM2319892_T116_LumB"
## [58] "GSM2319893_T120_Basal"  "GSM2319894_T121"        "GSM2319895_T122_Basal"
##  [61] "GSM2319896_T123_LumB"   "GSM2319897_T124_LumA"   "GSM2319898_T125_LumA"
##  [64] "GSM2319899_T126_LumB"   "GSM2319900_T129_LumA"   "GSM2319901_T135_LumA"
##  [67] "GSM2319902_T137_LumB"   "GSM2319903_T138_Normal" "GSM2319904_T140_Basal"
##  [70] "GSM2319905_T141_LumA"   "GSM2319906_T142_Normal" "GSM2319907_T143_Normal"
##  [73] "GSM2319908_T144_Basal"  "GSM2319909_T146_HER2"   "GSM2319910_T149_LumB"
##  [76] "GSM2319911_T150_LumB"   "GSM2319912_T154_LumA"   "GSM2319913_T155"
##  [79] "GSM2319914_T160_LumA"   "GSM2319915_T165_LumA"   "GSM2319916_T166_HER2"
##  [82] "GSM2319917_T168_Normal" "GSM2319918_T169_LumA"   "GSM2319919_T172_LumA"
##  [85] "GSM2319920_T174_Basal"  "GSM2319921_T175"        "GSM2319922_T176_LumB"
##  [88] "GSM2319923_T178_LumA"   "GSM2319924_T184_LumA"   "GSM2319925_T186_LumB"
##  [91] "GSM2319926_T187_LumA"   "GSM2319927_T189_Basal"  "GSM2319928_T190_LumA"
##  [94] "GSM2319929_T191_HER2"   "GSM2319930_T192_LumA"   "GSM2319931_T193_HER2"
##  [97] "GSM2319932_T194_LumA"   "GSM2319933_T196_LumB"   "GSM2319934_T198_Normal"
## [100] "GSM2319935_T200_LumA"
```

```
length(input)
```

```
## [1] 100
```

```
test <- input[1:5]
test
```

```
## GRangesList object of length 5:
## $GSM2319836_T1_Normal
## GRanges object with 204 ranges and 1 metadata column:
##         seqnames            ranges strand |     value
##            <Rle>         <IRanges>  <Rle> | <numeric>
##     [1]        1     61735-3614480      * |  1.384449
##     [2]        1   3614576-6784592      * |  0.943009
##     [3]        1   6792896-7248814      * |  1.343900
##     [4]        1  7251360-10502710      * |  2.538515
##     [5]        1 10505132-15172626      * |  1.357934
##     ...      ...               ...    ... .       ...
##   [200]       21 17459013-33927768      * |   2.30992
##   [201]       21 33927886-48096957      * |   1.83294
##   [202]       22 16052528-25705314      * |   1.56287
##   [203]       22 25706918-25910756      * |   3.95678
##   [204]       22 25910771-51234455      * |   1.46329
##   -------
##   seqinfo: 22 sequences from an unspecified genome; no seqlengths
##
## ...
## <4 more elements>
```

Run CINdex

standard run

Next, the first step of CINdex will start. This may take a week using core i5 10 gen while core i9 elevnt might take a 3-4 days. Do keep that in mind so that no laptop is used. If laptop is used for previous code, move towards a pc environment after importing the r data.

```
run.cin.chr(input)
```

The same warning as above, the CINdex package might take a long time. Using core i9 11th gen, almost 2 weeks where used to handle this command. It is recommended to run the command below using parallel processing if the computer can handle it. Please keep in mind also that the ram needed is arround 128 for parallel processing. ### run cytoband

```r
run.cin.cyto(grl.seg = input, cnvgr=cnvgr.19.auto, snpgr=snpgr.19.auto,
genome.ucsc = hg19.ucsctrack)
```

parallel processing code

```r
library(parallel)

# Define function to run the CINdex operation
run_cin_parallel <- function(V.mode, V.def, out.folder.name) {
  tryCatch({
    if (!dir.exists(out.folder.name)) {
      dir.create(out.folder.name, recursive = TRUE)
    }

    message("Running CINdex with mode: ", V.mode, " and V.def: ", V.def)

    # CINdex function
    run.cin.cyto(
      grl.seg = input,
      cnvgr = cnvgr.19.auto,
      snpgr = snpgr.19.auto,
      genome.ucsc = hg19.ucsctrack,
      V.def = V.def,
      V.mode = V.mode,
      chr.num = 22,
      out.folder.name = out.folder.name
    )

    message("Completed CINdex for mode: ", V.mode, " and V.def: ", V.def)
    return(TRUE)
  }, error = function(e) {
    message("Error in CINdex: ", e)
    return(FALSE)
  })
}

# Function to execute with a fixed number of cores (max CPU - 1)
run_with_fixed_cores <- function() {
  modes <- c("sum", "del", "amp")
  V.definitions <- list(normalized = 2, unnormalized = 3)

  # Use maximum available cores minus one
  n_cores <- detectCores() - 1
  message("Running with ", n_cores, " cores.")

  # Run all combinations of mode and normalization
  results <- mclapply(modes, function(mode) {
    lapply(names(V.definitions), function(normalization) {
      V.def <- V.definitions[[normalization]]
      folder_name <- paste0("output_", mode, "_", normalization)
      run_cin_parallel(V.mode = mode, V.def = V.def, out.folder.name = folder_name)
    })
  }, mc.cores = n_cores)

  return(results)
}

# Execute the function
results <- run_with_fixed_cores()
```

#Session info Thank you for reading. In you have any questions; results request of my run; and/or input for the code, please send to juan.lorell@student.i3l.ac.id.

```
sessionInfo()
```

```
## R version 4.4.2 (2024-10-31)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 24.04.1 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: Asia/Taipei
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] CINdex_1.34.0
##  [2] dplyr_1.1.4
##  [3] Homo.sapiens_1.3.1
##  [4] org.Hs.eg.db_3.20.0
##  [5] GO.db_3.20.0
##  [6] OrganismDbi_1.48.0
##  [7] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [8] GenomicFeatures_1.58.0
##  [9] AnnotationDbi_1.68.0
## [10] R.utils_2.12.3
## [11] R.oo_1.27.0
## [12] R.methodsS3_1.8.2
## [13] biovizBase_1.54.0
## [14] rtracklayer_1.66.0
## [15] pd.genomewidesnp.6_3.14.1
## [16] DBI_1.2.3
## [17] oligo_1.70.0
## [18] Biobase_2.66.0
## [19] oligoClasses_1.68.0
## [20] RSQLite_2.3.9
## [21] Biostrings_2.74.0
## [22] XVector_0.46.0
## [23] GenomicRanges_1.58.0
## [24] GenomeInfoDb_1.42.1
## [25] IRanges_2.40.1
## [26] S4Vectors_0.44.0
## [27] BiocGenerics_0.52.0
## [28] ASCAT_3.2.0
##
## loaded via a namespace (and not attached):
##   [1] RColorBrewer_1.1-3          rstudioapi_0.17.1
##   [3] jsonlite_1.8.9              magrittr_2.0.3
##   [5] rmarkdown_2.29              BiocIO_1.16.0
##   [7] zlibbioc_1.52.0             vctrs_0.6.5
##   [9] memoise_2.0.1               Rsamtools_2.22.0
##  [11] RCurl_1.98-1.16             base64enc_0.1-3
##  [13] progress_1.2.3              htmltools_0.5.8.1
##  [15] S4Arrays_1.6.0              curl_6.0.1
##  [17] SparseArray_1.6.0           Formula_1.2-5
##  [19] KernSmooth_2.23-24          htmlwidgets_1.6.4
##  [21] httr2_1.0.7                 cachem_1.1.0
##  [23] GenomicAlignments_1.42.0    lifecycle_1.0.4
##  [25] iterators_1.0.14            pkgconfig_2.0.3
##  [27] Matrix_1.7-1                R6_2.5.1
##  [29] fastmap_1.2.0               GenomeInfoDbData_1.2.13
##  [31] MatrixGenerics_1.18.0       digest_0.6.37
##  [33] colorspace_2.1-1            Hmisc_5.2-1
##  [35] filelock_1.0.3              fansi_1.0.6
##  [37] httr_1.4.7                  abind_1.4-8
##  [39] compiler_4.4.2              withr_3.0.2
##  [41] bit64_4.5.2                 doParallel_1.0.17
##  [43] htmlTable_2.4.3             backports_1.5.0
##  [45] BiocParallel_1.40.0         gplots_3.2.0
##  [47] biomaRt_2.62.0              rappdirs_0.3.3
##  [49] DelayedArray_0.32.0         rjson_0.2.23
##  [51] affxparser_1.78.0           caTools_1.18.3
##  [53] gtools_3.9.5                tools_4.4.2
##  [55] foreign_0.8-86              nnet_7.3-19
##  [57] glue_1.8.0                  restfulr_0.0.15
##  [59] grid_4.4.2                  checkmate_2.3.2
##  [61] cluster_2.1.6               generics_0.1.3
##  [63] gtable_0.3.6                BSgenome_1.74.0
##  [65] preprocessCore_1.68.0       ensembldb_2.30.0
##  [67] hms_1.1.3                   data.table_1.16.4
##  [69] xml2_1.3.6                  utf8_1.2.4
##  [71] foreach_1.5.2               pillar_1.9.0
##  [73] stringr_1.5.1               splines_4.4.2
##  [75] BiocFileCache_2.14.0        lattice_0.22-5
##  [77] bit_4.5.0.1                 RBGL_1.82.0
##  [79] tidyselect_1.2.1            knitr_1.49
##  [81] gridExtra_2.3               ProtGenerics_1.38.0
##  [83] SummarizedExperiment_1.36.0 xfun_0.49
##  [85] matrixStats_1.4.1          stringi_1.8.4
##  [87] UCSC.utils_1.2.0           lazyeval_0.2.2
##  [89] yaml_2.3.10                som_0.3-5.2
##  [91] evaluate_1.0.1             codetools_0.2-20
##  [93] tibble_3.2.1               graph_1.84.0
##  [95] BiocManager_1.30.25        cli_3.6.3
##  [97] affyio_1.76.0              rpart_4.1.23
##  [99] munsell_0.5.1              dichromat_2.0-0.1
## [101] dbplyr_2.5.0               png_0.1-8
## [103] XML_3.99-0.17              parallel_4.4.2
## [105] ggplot2_3.5.1              blob_1.2.4
## [107] prettyunits_1.2.0          AnnotationFilter_1.30.0
## [109] bitops_1.0-9               txdbmaker_1.2.1
## [111] ff_4.5.0                   VariantAnnotation_1.52.0
## [113] scales_1.3.0               crayon_1.5.3
## [115] rlang_1.1.4                KEGGREST_1.46.0
```