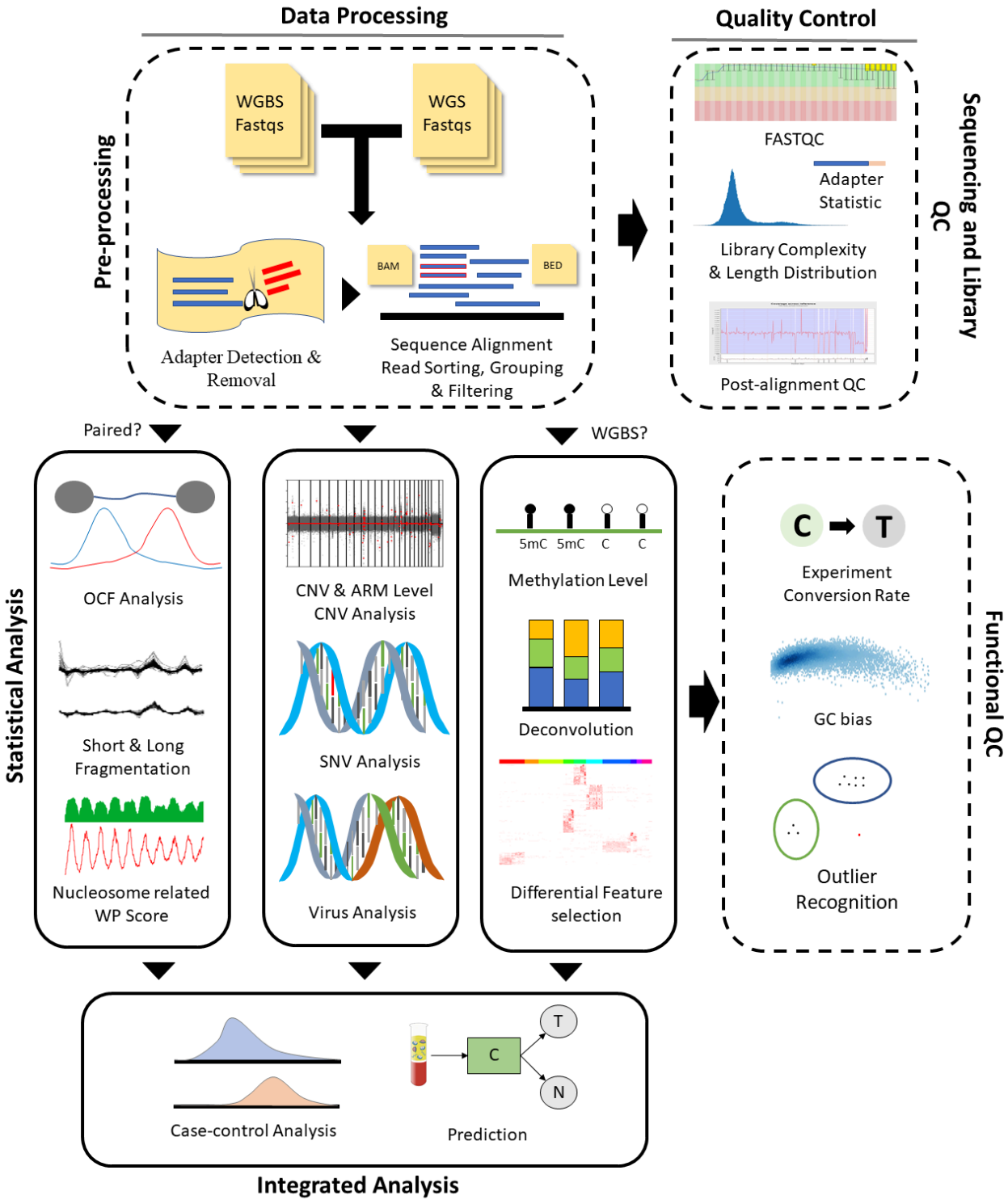
1 Introduction

cfDNApipe(Cell Free DNA Pipeline) is an integrated pipeline for analyzing cell-free DNA WGBS/WGS data. It contains many cfDNA quality control and feature extration algorithms. Also we collected some useful cell free DNA references and provide them here.

The whole pipeline was established based on flow graph principle. Users can use the integrated pipeline for WGBS/WGS data as well as build their own analysis pipeline from any intermediate data like bam files. The main functions are as the following picture.



cfDNApipe Functions

2 Package Installation and Loading

2.1 Download and Installation

The popular WGBS/WGS analysis software are released on Unix/Linux system, based on different program language, like Bowtie2 and Bismark. Therefore, it's very difficult to rewrite all the software in one language. Fortunately, conda/bioconda program collected many prevalent bioinformatics related python mudules and software, so we can install all the dependencies through conda/bioconda.If you did not install conda before, please follow this tutorial to install conda first.

After installation, you can create a new virtual environment for cfDNA analysis. Virtual environment management means that you can install all the dependencies in this virtual environment and delete them easily by removing this virtual environment.

2.2 Create Environment and Install Dependencies

We tested our pipeline using different version of software and provide an environment yml file for users. Users can download this file and create the environment in one command line without any software conflict.

first, please download the yml file.

```
wget https://raw.githubusercontent.com/Honchkrow/cfDNApipe/master/environment.yml
```

Then, run the following command to create an virtual environment named cfDNApipe. The environment will be created and all the dependencies as well as the latest cfDNApipe will be installed.

```
conda env create -n cfDNApipe -f environment.yml
```

Note: The environment name can be changed by replacing "-n cfDNApipe" to "-n environment_name".

2.3 Enter Environment and Use cfDNApipe

Once the environment is created, user can enter environment using the foloowing command.

```
conda activate cfDNApipe
```

Now, just open python and process **cell free DNA WGBS/WGS paired/single end** data.

3 Functional Summary

The whole workflow for cfDNApipe can be divided into two parts, raw data processing and statistical analysis.

3.1 Raw Data Processing and Quality Control

In this part, cfDNApipe offers functions to process raw sequencing data saved in fastq format to aligned bam or bed files. It wraps FASTQC for pre-alignment raw sequencing data quality control. AdapterRemoval is used for adapter detection and removal. Bowtie2 and Bismark are adopted for read alignment and BAM format output will be generated. In addition, cfDNApipe calls different duplication removal tools for WGS and WGBS data. The bam format output can be converted to BED format if needed. Post-alignment quality control will be executed by Qualimap, which reports basic information and statistics for the alignment data such as genome coverage and GC content. What's more, cfDNApipe can sort, index, group the aligned read for the downstream statistical analysis.

3.2 Statistical Analysis

In this part, cfDNApipe provides multiple state-of-the-art statistical analysis modules for cfDNA data. Methylation level can be culculated for any given genome regions and methylation signal will be deconvoluted to reveal the conponent changes in cfDNA. Large-scale CNV (arm-level) and small-scale CNV (bin-level) are culculated to reveal genomic gains and losses. cfDNA fragmentation analysis aims to demonstrate alters of DNA fragment length and disorder of fragmentation pattern. Orientation-aware
cfDNA fragmentation (OCF) analysis measures the differential phasing of upstream and downstream fragment ends in tissue-specific open chromatin regions. Besides, some functions are sepcific to cfDNA WGS data. Virus related to some specific diseases like HBV can be detected. Somatic and germline mutations is identified by comparing the sequence of DNA with that in control samples or default reference.

4 Quick Start for Preset Pipeline

cfDNApipe provides easy-to-use and friendly preset pipeline for cfDNA WGS/WGBS data. Users only need to provide the input raw sequencing files (FASTQ format), genome reference and output folder, then the program will do everything automatically. When the analysis finished, an pretty HTML report will be generated for demonstrating quality control and statistical analysis results. Here, we provide 2 examples for **single group analysis** and **case-control analysis**.

4.1 Single Group Analysis

Here, we show a single group analysis procedure for single end WGBS data. A runnable small dataset can be download from ***. All the samples should be put in a folder without any other files.

First load cfDNApipe and set global reference parameters for pipeline.

```
from cfDNApipe import *

pipeConfigure(
    threads=60,
    genome="hg19",
    refdir=r"reference_genome/hg19",
    outdir=r"output/single_WGBS",
    data="WGBS",
    type="single",
    build=True,
    JavaMem="10g",
)
```

Second, run the processing pipeline.

```
res = cfDNAWGBS(
    inputFolder=r"path_to_WGBS_SE",
    idAdapter=True,
    rmAdapter=True,
    rmAdOP={"--gzip": True},
    dudup=True,
    CNV=True,
    armCNV=True,
    fragProfile=True,
    verbose=True,
)
```

Now, you can wait for finish running.

The real dataset is from paper "Plasma DNA tissue mapping by genome-wide methylation sequencing for noninvasive prenatal, cancer, and transplantation assessments" with accession number EGAS00001001219 is processed. We only use Hepatocellular Carcinoma (HCC) patients data for demonstration. The final report can be downloaded from here.

4.2 Case Control Analysis

Here, we show a case controlanalysis procedure for paired end WGS data. A runnable small dataset can be download from ***. Each group samples should be put in each folder without any other files.

First load cfDNApipe and set global reference parameters for pipeline.

```
from cfDNApipe import *

pipeConfigure2(
    threads=60,
    genome="hg19",
    refdir="reference_genome/hg19",
    outdir="output/paired_WGS",
    data="WGS",
    type="paired",
    JavaMem="8G",
    case="cancer",
    ctrl="normal",
    build=True,
)
```

Second, run the processing pipeline.

```
a, b = cfDNAWGS2(
    caseFolder="path_to_data/case",
    ctrlFolder="path_to_data/ctrl",
    caseName="case",
    ctrlName="ctrl",
    idAdapter=True,
    rmAdapter=True,
    rmAdOP={"--gzip": True},
    bowtie2OP={"-q": True, "-N": 1, "--time": True},
    dudup=True,
    CNV=True,
    armCNV=True,
    fragProfile=True,
    OCF=True,
    report=True,
    verbose=False,
)
```

Now, you can wait for finish running.

The real dataset is from paper "Lengthening and shortening of plasma DNA in hepatocellular carcinoma patients" with accession number EGAS00001001024 is processed. We use Hepatocellular Carcinoma (HCC) patients and the healthy data for demonstration. The final report can be downloaded from here.

5 Customized Pipeline For Pipeline Function Verification

cfDNApipe integrates several state-of-the-art statistical models. Here, we illustrate how to use customized pipeline in cfDNApipe to perform these analysis.

The analyses performed below are start with bam or bed files. Some of them reproduce the results of the published papers.

5.1 Large-Scale CNV in HCC patient

CNV is a common phenomenon appears in kinds of cancer types. CNV is also detected from cfDNA WGS data reported by Jiang, et al. However, due to the different ctDNA cncentration, the aggregated signal in arm-level reveals more significant copy number changes.

Here, using the dataset from Jiang, et al, we will perform the Large-scale (arm-level) CNV to reveal the difference between HCC patients and the healthy.

First, set global parameters and get the aligned read files.

```
from cfDNApipe import *
import glob

pipeConfigure2(
    threads=20,
    genome="hg19",
    refdir=r"reference_genome/hg19",
    outdir=r"output/pcs_armCNV",
    data="WGS",
    type="paired",
    JavaMem="8G",
    case="cancer",
    ctrl="normal",
    build=True,
)

verbose = False

case_bam = glob.glob("path_to_data/HCC/*.bam")
ctrl_bam = glob.glob("path_to_data/CTR/*.bam")
```

Code above will generate the output folders for case and control based on the flag set in function "pipeConfigure2". The output folders are as below.

```
pcs_armCNV/
├──cancer/
│   ├── final_result/
│   ├── report_result/
│   └── intermediate_result/
└──normal/
    ├── final_result/
    ├── report_result/
    └── intermediate_result/
```

Second, compute GC corrected signal for case and control sample.

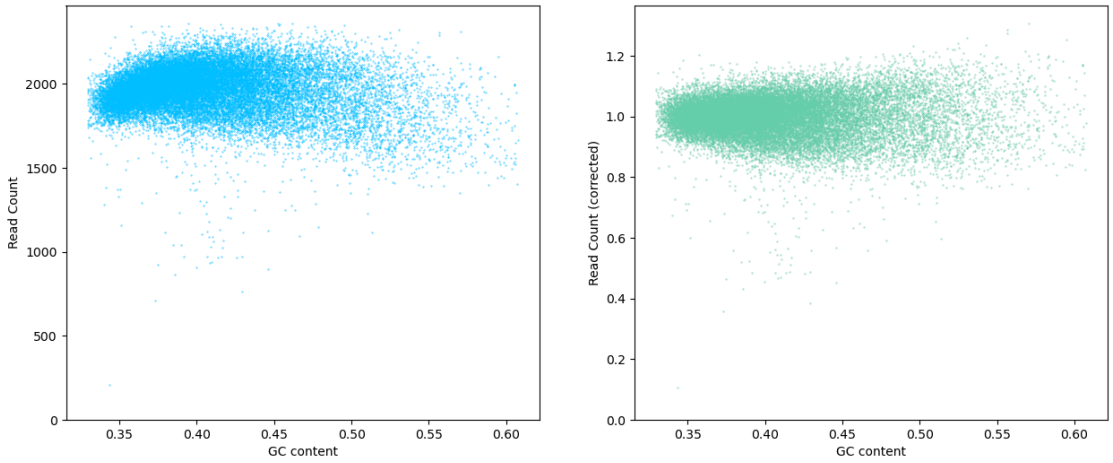
```
# count case
switchConfigure("cancer")
case_bamCounter = bamCounter(
    bamInput=case_bam, upstream=True, verbose=verbose, stepNum="case01"
)
case_gcCounter = runCounter(
    filetype=0, upstream=True, verbose=verbose, stepNum="case02"
)
case_GCCorrect = GCCorrect(
    readupstream=case_bamCounter,
    gcupstream=case_gcCounter,
    verbose=verbose,
    stepNum="case03",
)

# count ctrl
switchConfigure("normal")
ctrl_bamCounter = bamCounter(
    bamInput=ctrl_bam, upstream=True, verbose=verbose, stepNum="ctrl01"
)
ctrl_gcCounter = runCounter(
    filetype=0, upstream=True, verbose=verbose, stepNum="ctrl02"
)
ctrl_GCCorrect = GCCorrect(
    readupstream=ctrl_bamCounter,
    gcupstream=ctrl_gcCounter,
    verbose=verbose,
    stepNum="ctrl03",
)
```

After this step, users can find the outputs in each step folder like blew.

```
pcs_armCNV/
├──cancer/
│    ├── final_result/
│    ├── report_result/
│    └── intermediate_result/
│        ├── step_case01_bamCounter
│        ├── step_case02_runCounter
│        └── step_case03_GCCorrect
├──normal/
    ├── final_result/
    ├── report_result/
    └── intermediate_result/
        ├── step_ctrl01_bamCounter
        ├── step_ctrl02_runCounter
        └── step_ctrl03_GCCorrect
```

Users can also check the GC corrected results in folder with suffix "GCCorrect". HCC patient sample "H228" results is shown here.

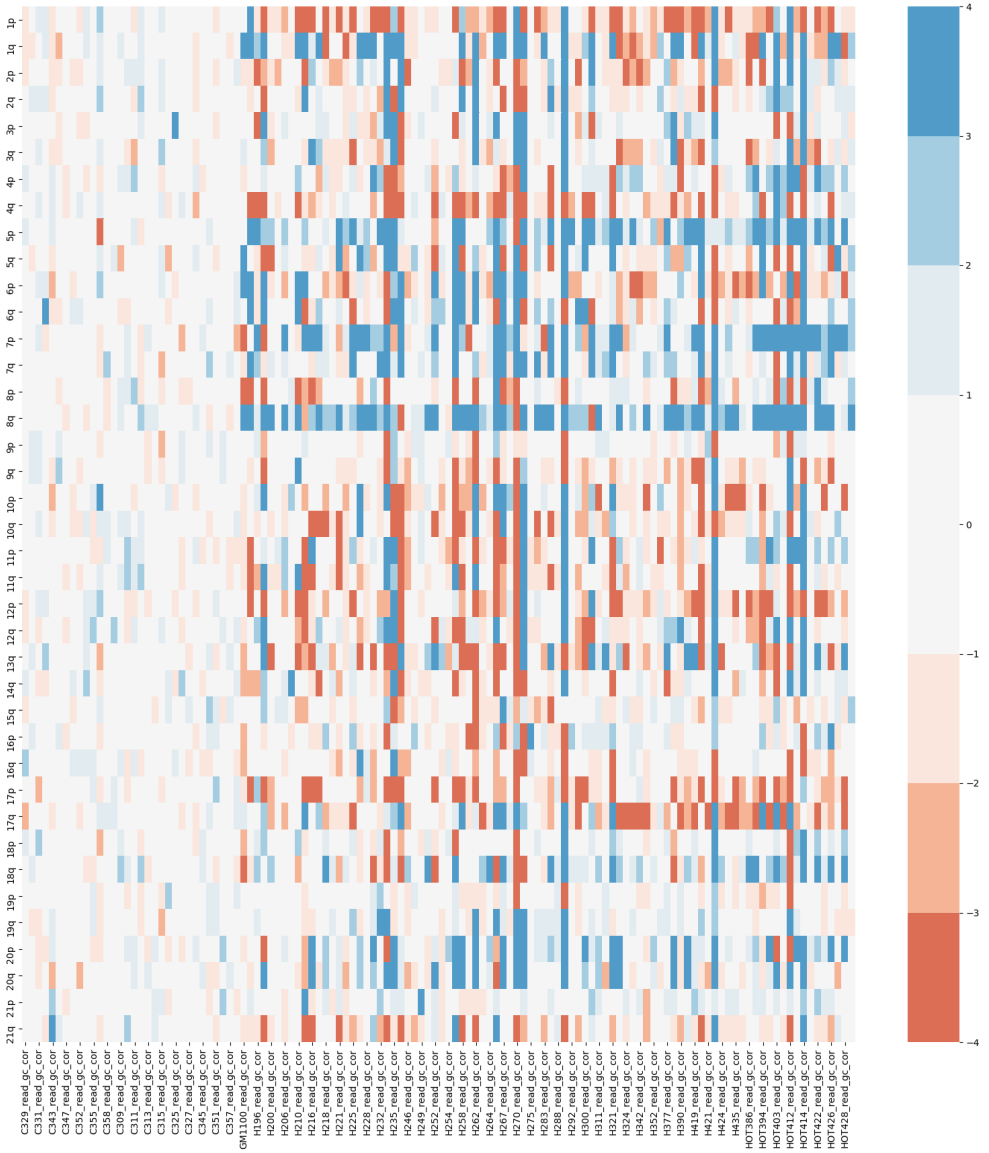


GC content correction for sample H228

Finally, compute Z-score for CNV signal and generate overall illustration for autosome.

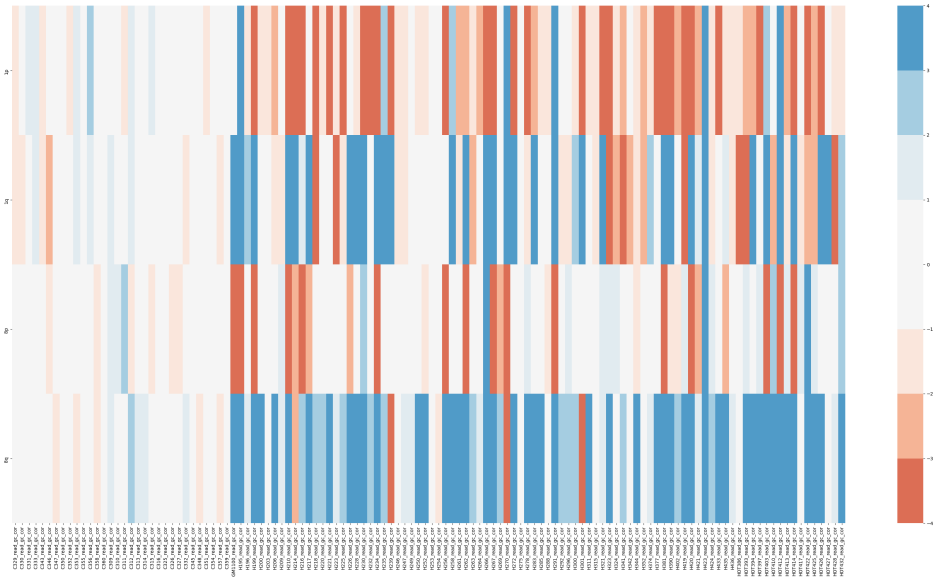
```
switchConfigure("cancer")
res_computeCNV = computeCNV(
    caseupstream=case_GCCorrect,
    ctrlupstream=ctrl_GCCorrect,
    stepNum="ARMCNV",
    verbose=verbose,
)
```

Users can find the final Z-score file and the figure for losses and gains like below.

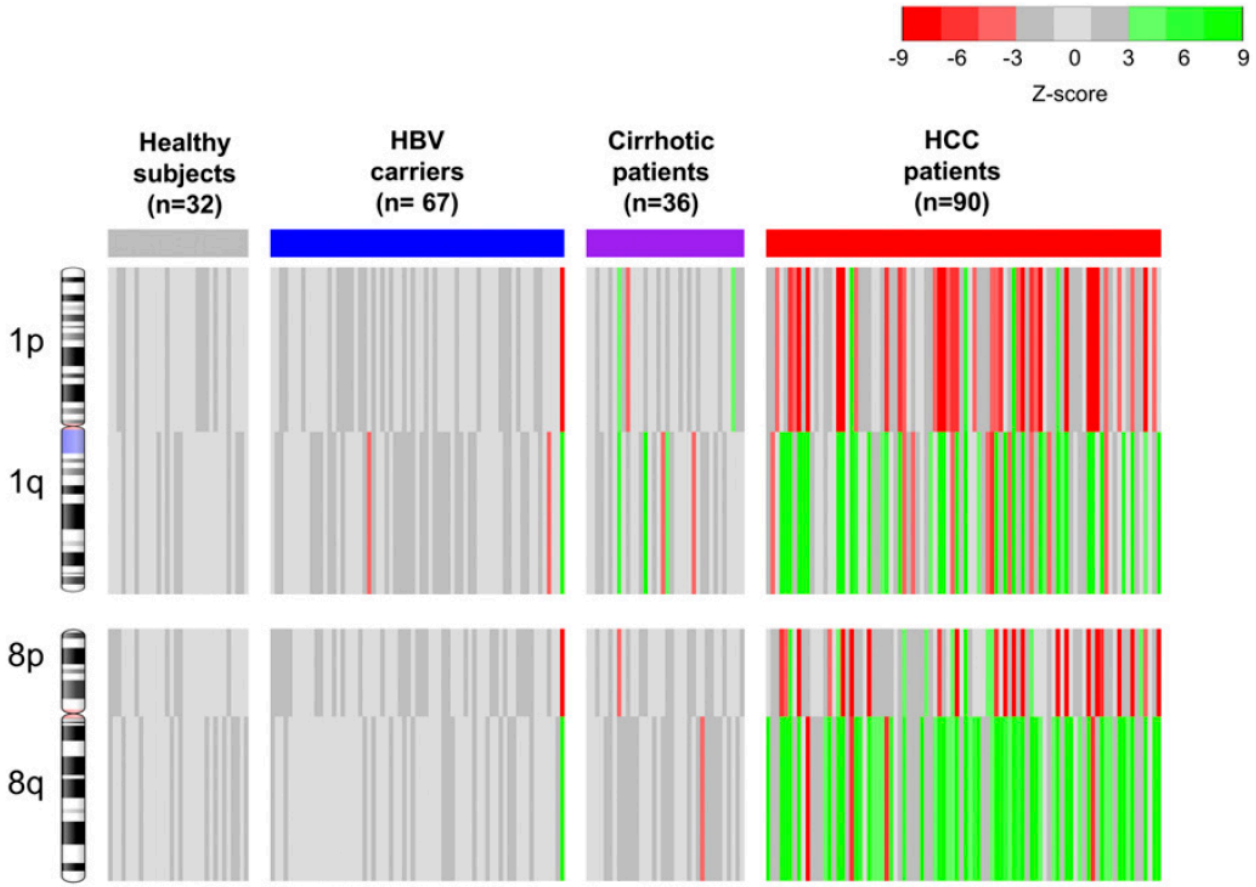


CNV Z-score for all samples

For HCC patients, the forper work has reported that p,q arm in chromatin 1 and 8 shows significant gain and loss. Ploting p,q arm in chromatin 1 and 8 in a new figure, we can see the similar results as the paper.



CNV Z-score in chromatin 1 and 8 for all samples



CNV Z-score in chromatin 1 and 8 for all samples from paper