PISA manual v0.12b

Shi Quan
shiquan@genomics.cn

April 25, 2022

Contents

The updated version of this document can be found at https://github.com/shiquan/PISA/blob/master/PISA_manual.pdf

1 Getting started.

1.1 Installation.

PISA source code can be downloaded at https://github.com/shiquan/PISA/release, or the development version from https://github.com/shiquan/PISA/. To compile PISA from sources run **make** in the source directory.

```
$ git clone https://github.com/shiquan/PISA
$ cd PISA
$ make
```

1.2 Quickly start.

The following test data can be found at `PISA/demo` directory.

```
$ cd demo

$ ls
aln.sam.gz barcodes.txt demo_1.fq.gz demo_2.fq.gz demo.gtf.gz peaks.bed README.md var.vcf.gz

$ zcat demo_1.fq.gz|head -n 4
@A00984:220:HNJ7KDRXX:1:1118:2510:4586
AAGCATCCACACAGAGCACCCCGTTCTT
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFF

$ zcat demo_2.fq.gz|head -n 4
@A00984:220:HNJ7KDRXX:1:1118:2510:4586
GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
+
FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF

# Convert raw FASTQ to FASTQ+ format
$ PISA parse2 -rule 'CR,R1:1-16,barcodes.txt,CB,1;UR,R1:17-28;R1,R2' demo_1.fq.gz demo_2.fq.gz -1
    demo.fq
Number of Fragments,825
Fragments pass QC,825
Fragments with Exactly Matched Barcodes,805
Fragments with Failed Barcodes,0
[2022-04-22 12:21:35] Real time: 0.003 sec; CPU: 0.009 sec; Peak RSS: 0.010 GB.

$ head -n 4 demo.fq
@A00984:220:HNJ7KDRXX:1:1118:2510:4586|||CR:Z:AAGCATCCACACAGAG|||CB:Z:AAGCATCCACACAGAG|||UR:Z:CACCCCGTTCTT
GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
+
FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF

# Alignment results of FASTQ+
$ samtools view aln.sam.gz|head -n 1
A00984:220:HNJ7KDRXX:1:1118:2510:4586|||CR:Z:AAGCATCCACACAGAG|||CB:Z:AAGCATCCACACAGAG|||UR:Z:CACCCCGTTCTT
    0 chr11  35139165      255    26S65M *       0       0
    GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
    FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF
    NH:i:1 HI:i:1 AS:i:61 nM:i:1
```

```
# Convert format alignment records from SAM to BAM
$ PISA sam2bam aln.sam.gz -o aln.bam
Raw reads,825
Mapped reads,820 (99.39%)
Plus strand,820
Minus strand,0
Mitochondria ratio,0.00%
[2022-04-22 12:26:31] Real time: 0.005 sec; CPU: 0.009 sec; Peak RSS: 0.010 GB.

$ samtools view aln.bam|head -n 1
A00984:220:HNJ7KDRXX:1:1118:2510:4586 0    chr11  35139165      255    26S65M *      0
    0
    GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
    FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF
    NH:i:1 HI:i:1 AS:i:61 nM:i:1 CR:Z:AAGCATCCACACAGAG CB:Z:AAGCATCCACACAGAG UR:Z:CACCCCGTTCTT

# Annotate gene names for BAM
$ PISA anno -gtf ./demo.gtf.gz aln.bam -o anno_gtf.bam
[2022-04-22 12:28:38] GTF loading..
[2022-04-22 12:28:38] Load 2 genes.
[2022-04-22 12:28:38] Load time : 0.003 sec
Reads Mapped to Genome (Map Quality >= 0),99.4%
Reads Mapped to Exonic Regions,99.3%
Reads Mapped to Intronic Regions,0.0%
Reads Mapped to both Exonic and Intronic Regions,0.7%
Reads Mapped Antisense to Gene,0.0%
Reads Mapped to Intergenic Regions,0.0%
Reads Mapped to Gene but Failed to Interpret Type,0.0%
[2022-04-22 12:28:38] Real time: 0.026 sec; CPU: 0.086 sec; Speed : 9528 records/sec; Peak RSS:
    0.034 GB.

# Correct UMIs amongst other UMIs from the same cell and mapped to the same gene, and create new
    tag UB for corrected UMIs
$ PISA corr -tag UR -new-tag UB -tags-block CB,GN anno_gtf.bam -o corr.bam
[2022-04-22 12:36:21] Building index ..
[2022-04-22 12:36:21] Build time : 0.002 sec
[2022-04-22 12:36:21] Real time: 0.077 sec; CPU: 0.085 sec

$ samtools view corr.bam|head -n 1
A00984:220:HNJ7KDRXX:1:1118:2510:4586 0    chr11  35139165      255    26S65M *      0
    0
    GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
    FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF
    NH:i:1 HI:i:1 AS:i:61 nM:i:1 CR:Z:AAGCATCCACACAGAG CB:Z:AAGCATCCACACAGAG UR:Z:CACCCCGTTCTT
    RE:A:E GX:Z:ENSG00000026508.18 GN:Z:CD44
    TX:Z:ENST00000263398.10,ENST00000428726.7,ENST00000526025.2 UB:Z:CACCCCGTTCTT

# Count gene X cell features
$ mkdir exp

$ PISA count -cb CB -anno-tag GN -outdir exp -umi UB corr.bam
[2022-04-22 12:38:44] Real time: 0.033 sec; CPU: 0.013 sec; Peak RSS: 0.010 GB.

# Gene expression matrix generated in the Market Exchange format
$ ls exp/
barcodes.tsv.gz features.tsv.gz matrix.mtx.gz

# Not just gene features, we can also annotate variants and functional regions to reads
```

```
$ PISA anno -bed peaks.bed -tag PK -vcf var.vcf.gz -vtag VR corr.bam -o anno_vcf_bed.bam
Reads Mapped to Genome (Map Quality >= 0),99.4%
Reads Mapped to BED regions / Peaks,0.0%
[2022-04-22 12:43:01] Real time: 0.027 sec; CPU: 0.090 sec; Speed : 9085 records/sec; Peak RSS:
    0.034 GB.

$ samtools view anno_vcf_bed.bam|grep "VR:Z"|grep "PK:Z"|head -n 1
A00984:220:HNJ7KDRXX:1:2266:27597:30843 0   chr11   35229688      255     91M     *       0
    0
    CCCAGGGTTAATAGGGCCTGGTCCCTGGGAGGAAATTTGAATGGGTCCATTTTGCCCTTCCATAGCCTAATCCCGGGGCATTGTTTTCCAC
FFFF,FFFFFFFFFF,FFFFFFF:,:FFFFFF,FF:FFFFFFFFFF:,,:F:F:F,F:F:F,FFFF,F:FFFF,FF,FF:FFFFFFF:F: NH:i:1
    HI:i:1 AS:i:85 nM:i:2 CR:Z:ATTGTTCCAAGTCCCG CB:Z:ATTGTTCCAAGTCCCG UR:Z:TCTTTAAGTCAG RE:A:E
    GX:Z:ENSG00000026508.18 GN:Z:CD44
    TX:Z:ENST00000263398.10,ENST00000428726.7,ENST00000425428.6,ENST00000433892.6,ENST00000525469.1
    UB:Z:TCTTTAAGTCAG PK:Z:demo_peak_14a;demo_peak_14b VR:Z:chr11_35229771_C_T

# Summarize the reads, UMIs, genes, peaks, and variants per cell
$ PISA attrcnt -cb CB -tags UB,GN,PK,VR anno_vcf_bed.bam -dedup |head -n 5
BARCODE Raw    UB      GN      PK      VR
AAGCATCCACACAGAG      503     132     1       15      1
ATTGTTCCAAGTCCCG      533     124     1       13      1
AAGCATCCACACNGAG      3       1       1       1       0
AAGCNTCCACACAGAG      3       1       1       1       0

# Deduplicate BAM file for each cell
$ PISA rmdup -tags CB corr.bam -o rmdup.bam -nw
[2022-04-22 12:59:39] Deduplicating chr11
[2022-04-22 12:59:39] All reads,820
[2022-04-22 12:59:39] Duplicate reads,125
[2022-04-22 12:59:39] Duplicate ratio,0.1524
[2022-04-22 12:59:39] Real time: 0.008 sec; CPU: 0.015 sec; Peak RSS: 0.010 GB.

# Deduplicate BAM file for each molecular
$ PISA rmdup -tags CB,UR corr.bam -o rmdup1.bam -nw
[2022-04-22 13:00:35] Deduplicating chr11
[2022-04-22 13:00:35] All reads,820
[2022-04-22 13:00:35] Duplicate reads,0
[2022-04-22 13:00:35] Duplicate ratio,0.0000
[2022-04-22 13:00:35] Real time: 0.009 sec; CPU: 0.015 sec; Peak RSS: 0.011 GB.

# Select all reads annotated to gene CD44
# Generate a gene candidate list
$  echo "CD44" > gene.txt

$ PISA pick -tags GN -list gene.txt anno_vcf_bed.bam -o picked.bam
[2022-04-22 13:03:01] Real time: 0.009 sec; CPU: 0.016 sec

# Select reads with more features
$ awk '{printf("%s\tCD44\n", $1)}' barcodes.txt > candidates.txt

$ cat candidates.txt
AAGCATCCACACAGAG      CD44
ATTGTTCCAAGTCCCG      CD44
GCACATAGTCAGTTTG      CD44

$ PISA pick -tags CB,GN -list candidates.txt anno_vcf_bed.bam -o picked.bam
[2022-04-22 13:09:28] Real time: 0.007 sec; CPU: 0.013 sec

# Convert BAM to FASTQ+
```

```
$ PISA bam2fq -tags CB,GN picked.bam -o pick.fq

$ head -n 4 pick.fq
@A00984:220:HNJ7KDRXX:1:1118:2510:4586|||CB:Z:AAGCATCCACACAGAG|||GN:Z:CD44
GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
+
FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF

# Sort FASTQ+ reads by CB and GN
$ PISA fsort -tags CB,GN pick.fq -o fsort.fq.gz
[2022-04-22 13:22:50] Write 795 records to fsort.fq.gz.0000.bgz.
[2022-04-22 13:22:50] Unlink fsort.fq.gz.0000.bgz
[2022-04-22 13:22:50] Create fsort.fq.gz from 1 files.
[2022-04-22 13:22:50] Real time: 0.021 sec; CPU: 0.020 sec

$ zcat fsort.fq.gz|head -n 8
@A00984:220:HNJ7KDRXX:1:1118:2510:4586|||CB:Z:AAGCATCCACACAGAG|||GN:Z:CD44
GCAGTGGTATCAACGCAGAGTACATGGGGAGCCTCATTGCCCAGCGGACCCCAGCCTCTGCCAGGTTCGGTCCGCCATCCTCGTCCCGTCC
+
FFFFF:FFFFFF:FFFFFFFFFF,FFFFFFFFF:FFFFFFFFF,FFFF:FFF,FFFFF:FFF:FFFFFFFFFFFFFFFFFFFF:FFFFFFF
@A00984:220:HNJ7KDRXX:1:2143:21640:21496|||CB:Z:AAGCATCCACACAGAG|||GN:Z:CD44
CCTGCCCCGCGCCCAGAGATCCTCCAGCTCCTTTCGCCCGCGCCCTACGTTCGCTCCGGACACCATGGACAAGTATTGGTGGAACACAGCC
+
,,FFFFFFFF,F,,:F,F,F:FFF,FFFFFF,F::F,FF,F:FF:,,FF:FFF,:FFFF:FFF,::FFFF,F:F,FFFF,,F,FFF,F,::

# Assembly reads mapped to CD44 of the same cell into contigs
# This step requires Trinity software and seqtk already installed in your environment
$ PISA stream -tags CB,GN -script 'Trinity --seqType fq --SS_lib_type F --single ${FQ}
    --max_memory 1G 2>/dev/null 1>/dev/null; seqtk rename trinity_out_dir.Trinity.fasta ${UBI}_
    2>/dev/null' -t 10 -fa -nw ./fsort.fq.gz -o assem.fa
[2022-04-22 13:24:21] Real time: 5.607 sec; CPU: 0.010 sec; Peak RSS: 0.010 GB.

$ seqtk seq assem.fa -l 100 |head
>Z_AAGCATCCACACAGAG_Z_CD44_1|||CB:Z:AAGCATCCACACAGAG|||GN:Z:CD44 len=439 path=[0:0-438]
GAAATTAGGGCCCAATTAATAATCAGCAAGAATTTGATCGTTCCAGTTCCCACTTGGAGGCCTTTCATCCCTCGGGTGTGCTATGGATGGCTTCTAACAA
AAACTACACATATGTATTCCTGATCGCCAACCTTTCCCCCACCAGCTAAGGACATTTCCCAGGGTTAATAGGGCCTGGTCCCTGGGAGGAAATTTGAATG
GGTCCATTTTGCCCTTCCATAGCCTAATCCCTGGGCATTGTTTTCCACTGAGGTTGGGGGTTGGGGTGTACTAGTTACACATCTTCAACAGACCCCCTCT
AGAAATTTTTCAGATGCTTCTGGGAGACACCCAAAGGGTGAAGCTATTTATCTGTAGTAAACTATTTATCTGTGTTTTTGAAATATTAAACCCTGGATCA
GTCCTTTGATCAGTATAATTTTTTAAAGTTACTTTGTCA
>Z_AAGCATCCACACAGAG_Z_CD44_2|||CB:Z:AAGCATCCACACAGAG|||GN:Z:CD44 len=384 path=[0:0-383]
CCTGGTAGAATTGGCTTTTCTAGCAGAACCTTTCCAAAAGTTTTATATTGAGATTCATAACAACACCAAGAATTGATTTTGTAGCCAACATTCATTCAAT
ACTGTTATATCAGAGGAGTAGGAGAGAGGAAACATTTGACTTATCTGGAAAAGCAAAATGTACTTAAGAATAAGAATAACATGGTCCATTCACCTTTATG
TTATAGATATGTCTTTGTGTAAATCATTTGTTTTGAGTTTTCAAAGAATAGCCCATTGTTCATTCTTGTGCTGTACAATGACCACTGTTATTGTTACTTT

# Align assembled reads to reference and convert to BAM file
# Here I use minimap2 for simplicity
$ minimap2 -x splice -a ~/Documents/datasets/GRCh38/fasta/genome.fasta assem.fa 1> asm_aln.sam
[M::mm_idx_gen::50.495*1.81] collected minimizers
[M::mm_idx_gen::71.980*2.16] sorted minimizers
[M::main::71.980*2.16] loaded/built the index for 194 target sequence(s)
[M::mm_mapopt_update::75.057*2.11] mid_occ = 767
[M::mm_idx_stat] kmer size: 15; skip: 5; is_hpc: 0; #seq: 194
[M::mm_idx_stat::77.004*2.09] distinct minimizers: 167225302 (35.42% are singletons); average
    occurrences: 6.036; average spacing: 3.071; total length: 3099750718
[M::worker_pipeline::77.010*2.09] mapped 13 sequences
[M::main] Version: 2.21-r1071
[M::main] CMD: minimap2 -x splice -a /home/shiquan/Documents/datasets/GRCh38/fasta/genome.fasta
    assem.fa
[M::main] Real time: 77.358 sec; CPU: 161.000 sec; Peak RSS: 18.519 GB
```

```
$ PISA sam2bam asm_aln.sam -o asm_aln.bam
Raw reads,13
Mapped reads,13 (100.00%)
Plus strand,13
Minus strand,0
Mitochondria ratio,0.00%
[2022-04-22 13:30:20] Real time: 0.001 sec; CPU: 0.006 sec; Peak RSS: 0.010 GB.

$ samtools view asm_aln.bam|head -n 1
Z_AAGCATCCACACAGAG_Z_CD44_1  0      chr11  35229531      60     439M   *      0      0
    GAAATTAGGGCCCAATTAATAATCAGCAAGAATTTGATCGTTCCAGTTCCCACTTGGAGGCCTTTCATCCCTCGGGTGTGCTATGGATGGCTTCTAACAA
AAACTACACATATGTATTCCTGATCGCCAACCTTTCCCCCACCAGCTAAGGACATTTCCCAGGGTTAATAGGGCCTGGTCCCTGGGAGGAAATTTGAATG
GGTCCATTTTGCCCTTCCATAGCCTAATCCCTGGGCATTGTTTTCCACTGAGGTTGGGGGTTGGGGTGTACTAGTTACACATCTTCAACAGACCCCCTCT
AGAAATTTTTCAGATGCTTCTGGGAGACACCCAAAGGGTGAAGCTATTTATCTGTAGTAAACTATTTATCTGTGTTTTTGAAATATTAAACCCTGGATCA
GTCCTTTGATCAGTATAATTTTTTAAAGTTACTTTGTCA *  NM:i:1 ms:i:436      AS:i:436      nn:i:0 tp:A:P
    cm:i:137      s1:i:430      s2:i:0 de:f:0.0023   rl:i:0 CB:Z:AAGCATCCACACAGAG GN:Z:CD44
```

2 Description of all tools and options.

2.1 parse2

The `parse` (will be introduce in next subsection) and `parse2` tools are designed to convert FASTQ to FASTQ+ files. The `parse2` uses -rule option to describe the library structure. We also implement the predefined common library structures with the release, `-x` option can be used to specify the code. This tool is faster than the `parse` tool but only supports correct barcodes within 1 mismatch.

```
# Parse cell barcode and UMI string from raw FASTQ.

$ PISA parse2 -rule CB,R1:1-10,whitelist.txt,CB,1;R1,R1:11-60;R2,R2 -report fastq.csv \
         lane1_1.fq.gz,lane02_1.fq.gz lane1_2.fq.gz,lane2_2.fq.gz

Options :
 -1      [fastq]  Read 1 output.
 -2      [fastq]  Read 2 output.
 -rule   [STRING] Read structure in line. See Notice.
 -p               Read 1 and read 2 interleaved in the input file.
 -q      [INT]    Drop reads if average sequencing quality below this value.
 -dropN           Drop reads if N base in sequence or barcode.
 -report [csv]    Summary report.
 -t      [INT]    Threads. [4]
 -order           Keep input order.
 -x               Predefined code for specific library.
        * C4      Library structure for DNBelab C4 RNA kit v1.
Notice :
 * -rule accept tag rule STRING to parse input fastq following format
     "TAG,location,whitelist,corrected TAG,allow mismatch".
   For each tag rule, location part should be format like R[12]:start-end. Both start and end
       location start from 1.
   TAG and locaion parts are mandatory, and whitelist, corrected TAG and mismatch are optional.
   Futhermore, multiply tags seperated by ';'. In location part, R1 stands for raw read 1, R2
       stands for raw read 2.
   In tag part, R1 stands for output read 1 while R2 stands for output read 2. Here are some
       examples.

$ PISA parse2 -rule 'CR,R1:1-18,barcodes.txt,CB,1;UR,R1:19-30;R1,R2:1-100' -1 read_1.fq
    raw_read_1.fq raw_read_2.fq
```

```
# CR,R1:1-18,barcodes.txt,CB,1 - CR tag start from 1 to 18 in read 1, and barcodes.txt are barcode
    whitelist,
#   each barcode per line. Cell barcode will be corrected while hamming distance <= 1.
#   Corrected cell barcode tag is CB.
# UR,R1:19-30 - UR tag start from 19-30 in read 1.
# R1,R2:1-100 - Sequence from 1 to 100 in read 2 output to read 1 file.

$ PISA parse2 -rule 'CR,R1:1-10,bc1.txt,CB,1;CR,R1:11-20,bc2.txt,CB,1;R1,R2:1-100' -1 read_1.fq
    raw_read_1.fq raw_read_2.fq

# CR,R1:1-10,bc1.txt,CB,1;CR,R1:11-20,bc2.txt,CB,1 - This cell barcode consist of two segments,
    first segment start
#   from 1 to 10 in read 1, and whitelist is bc1.txt, and second segment start from 11 to 20, and
    whitelist is bc2.txt.
#   These two segments will be combined after correction, because the corrected tag are the same.
```

Option `-report` can specify a quality control report in CSV format. Here is the explanation of each term in this file.

Terms	Description
Number of Fragments	The number of records in the FASTQ(s). For paired reads, each pair only count once.
Fragments pass QC	Reads or paired reads pass QC.
Fragments with Exactly Matched Barcodes	Barcodes exactly matched with any barcode in the candidate list.
Fragments with Failed Barcodes	No barcode found in the candidate list with similar search.

2.2 parse

The `parse` tool requires a configure file to describe the library structure, including cell barcodes locations, UMI locations, and tag names. In addition, it will generate various quality control reports to stdout and cell barcodes distribution. The complete options list is below.

```
$ PISA parse -config read_struct.json -report fastq.csv -cbdis cell_dist.tsv \
        -1 out.fq lane1_1.fq.gz,lane02_1.fq.gz lane1_2.fq.gz,lane2_2.fq.gz

Options :
 -1      [fastq]  Read 1 output. Default is stdout.
 -2      [fastq]  Read 2 output.
 -config [json]   Read structure configure file in JSON format. Required.
 -run    [string] Run code, used for different library.
 -cbdis  [FILE]   Read count per cell barcode.
 -p               Read 1 and read 2 interleaved in the input file.
 -q      [INT]    Drop reads if average sequencing quality below this value.
 -dropN           Drop reads if N base in sequence or barcode.
 -report [csv]    Summary report.
 -t      [INT]    Threads. [4]
```

`-config` requires a JSON file to tell software the locations of cell barcodes and/or UMIs. An example configured file can be found at demo/demo.json.

```
{
    "cell barcode tag":"CB",
    "cell barcode":[
```

```
    {
        "location":"R1:1-16" # the location is 1 based
    }
    ],
    "UMI tag":"UR",
    "UMI":{
    "location":"R1:17-28",
    },
    "read 1":{
    "location":"R2:1-91",
    }
}
```

Option `-report` can specify a quality control report in CSV format. Here is the explanation of each term in this file.

Terms	Description
Number of Fragments	The number of records in the FASTQ(s). For paired reads, each pair only count once.
Fragments pass QC	Reads or paired reads pass QC.
Fragments with Exactly Matched Barcodes	Cell barcode exactly matched with any barcode in the candidate list.
Fragments with Failed Barcodes	No cell barcode found in the candidate list with similar search.
Fragments Filtered on Low Quality	Mean sequence quality of the records smaller than threshold.
Fragments Filtered on Unknown Sample Barcodes	Sample barcodes not matched with any barcode in the candidate list.
Q30 bases in Cell Barcode	Ratio of bases in cell barcode sequence with quality >= 30.
Q30 bases in Sample Barcode	Ratio of bases in sample barcode sequence with quality >= 30.
Q30 bases in UMI	Ratio of bases in UMI sequence with quality >= 30.
Q30 bases in Reads	Ratio of bases in read sequence with quality >= 30.

Here is an example to parse barcodes and reads with a predefined configure JSON file, the test files can be found at demo directory.

```
$ PISA parse -config demo/demo.json -report demo/parse.csv demo/demo_1.fq.gz demo/demo_2.fq.gz >
    demo/demo.fq
```

2.3 fsort

The `fsort` tool is designed to order the FASTQ+ records by tags. And option `-tags` can specify these tags. In general, this tool directly sorts a file if the size is smaller than 1 gigabyte. But for a large FASTQ, caching the whole file in the memory is not applicable, so this tool will split a large file into small pieces, sort each piece individually and merge the sorted records.

```
# Sort reads by tags.

$ PISA fsort -tags CB,UR -list cell_barcodes_top10K.txt -@ 5 -o sorted.fq.gz in.fq

Options:
 -tags    [TAGS]    Tags, such as CB,UR. Order of these tags is sensitive.
 -@       [INT]     Threads to compress file.
 -o       [fq.gz]   bgzipped output fastq file.
 -m       [mem]     Memory per thread. [1G]
 -p                 Input fastq is smart pairing.
```

```
 -T      [prefix]  Write temporary files to PREFIX.nnnn.tmp
```

2.4 stream

For the sorted FASTQ+ file, we defined the FASTQ+ block, which has the same tags and is grouped in the file. The `stream` tool is actually a framework to split each FASTQ+ block into a 'small' FASTQ+ file. The -script specifies a user-defined bash script. This script will read the small FASTQ+ file and generate a FASTQ/FASTA output to stdout. Then the stream tool will collect the stdout by pipe and update the tags to make sure each block of reads still keeps the original tags. And all the output into a file. In summary, this tool divides FASTQ+ blocks and conquers each of them with a user-defined process and merges results.

```
# Perform user-defined script for each FASTQ+ block.

$ PISA stream -script run.sh reads.fq.gz

Options :
 -tags   [TAGS]    Tags to define read blocks.
 -script [FILE]    User defined bash script, process $FQ and generate results to stdout.
 -min    [INT]     Mininal reads per block to process. [2]
 -keep             Output unprocessed FASTQ+ records.
 -fa               Stream FASTQ output instead of FASTQ.
 -tmpdir
 -t                Threads.
 -o      [FILE]    Path to output file.
 -nw               Disable warning messages.
```

2.4.1 Write a script for PISA stream

`PISA stream` will create a file `"_block.fq"` for each block of reads in the temporary directory. And the path of this file will store in the environment parameter of the subprocess as `"${FQ}"`. Furthermore, to determine the uniqueness of each block, the alias name, called the unique block index, will also export to the environment as `${UBI}`. The following script converts the FASTQ+ to FASTA and renames sequence id. Users should remember the last step of this script should also export a FASTQ+ or FASTA to stdout. And besides the last step, other steps should not generate any information to stdout and stderr. Because PISA uses to open the script's output by pipe, unschedule characters will break the format. The script could write in a bash file or inline in the command.

```
seqtk rename ${FQ} > test.fa; seqtk rename test.fa ${UBI}
```

2.5 addtags

The `addtags` tool is designed to put the new tags or update tags for FASTQ+ or BAM files.

```
# Just add tags to reads.

$ PISA addtags -str CB:Z:CELL1,LB:Z:PoII2 -o out.bam in.bam
$ PISA addtags -str CB:Z:CELL1,LB:Z:PoII2 -o out.fastq in.fastq

Options:
-o    [FILE]   Output file, bam or fastq, depends on input format
-str  [string] TAGs.
-@    [INT]    Threads to pack file.
-mapq [INT]    Mapping quality score to filter mapped reads.
```

2.6 sam2bam

After alignment, the sequence id of FASTQ+ records will keep in the RNAME of the SAM file. Considering the RNAME has been limited to 254 bits, we also limit the sequence id and the optional tag fields in FASTQ+ to this length. The sam2bam tool will parse the tags from the RNAME and put these tags to the end of SAM optional fields.

```
# Parse FASTQ+ read name and convert SAM to BAM.

$ PISA sam2bam -report alignment.csv -@ 5 -adjust-mapq -gtf genes.gtf -o aln.bam in.sam[.gz]

Options :
 -o      [BAM]      Output file [stdout].
 -t      [INT]      Work threads.
 -mito   [string]   Mitochondria name. Used to stat ratio of mitochondria reads.
 -maln   [BAM]      Export mitochondria reads into this file instead of standard output file.
 -@      [INT]      Threads to compress bam file.
 -report [csv]      Alignment report.

Note :
* Reads map to multiple loci usually be marked as low quality and filtered at downstream analysis.
  But for RNAseq library, if reads map to an exonic locus but also align to 1 or more non-exonic
      loci,
  the exonic locus can be prioritized as primary alignments, and mapping quality adjusts to 255.
      Tag
  MM:i:1 will also be added for this record. Following options used to adjust mapping quality.
* Input SAM need be sorted by read name, and aligner should output all hits of a read in this SAM.
 -adjust-mapq       Enable adjusts mapping quality score.
 -gtf    [GTF]      GTF annotation file. This file is required to check the exonic regions.
 -qual   [255]      Updated quality score.
```

Option -t is to set the threads to parse the SAM records. And `-@` option is to set the threads to compress alignments in BGZF format. The default compress level of BGZF is 6 in the htslib, but here PISA has reset this value to 2 to decrease the CPU times. But this will also increase the size of the BAM file. If users want to store the BAM file for a long time, converting the processed BAM files to CRAM format is recommended. The `-t` and `-@` options will be merged in the future.

Option `-report` can specify a quality control report in CSV format. Here is the explanation of each term in this file.

Terms	Description
Raw reads	Raw reads in the BAM files, secondary alignment will be skipped.
Mapped reads	Reads mapped to reference, and the ratio of raw reads.
Plus strand	Reads mapped to forward strand of reference.
Minus strand	Reads mapped to backward strand of reference.
Mitochondria ratio	Ratio of reads mapped to chromosome mitochondria. The default mito name is "chrM", user should change it by `-mito` option if reference is different. Otherwise this value will always be 0.

For RNA library, a read from cDNA map to an exonic locus but also map to one or more non-exonic regions, the exonic locus can be prioritized as primary alignments, and mapping quality adjusts to 255. In the below records, read DP8400008965TLL1C001R0102043364 mapped to three loci, and the aligner random pick one as primary alignment and others are secondary. Each of these alignments has low mapping quality (MAPQ == 2, usually be filtered at downstream analysis). Our adjustment method will check if only one of these alignments overlaps with exonic regions. In our example, the last alignment overlapped with gene EEF1A1, and the other two hit intergenic regions. After adjustment, the last record has been flag as a primary hit, and the mapping quality adjusts to 255, other alignments of the same read are updated as

secondary, MAPQ adjust to 0. MM:i:1 tag also is added to the primary record after adjustment. Option `-adjust-mapq` is inspired by 10X CellRanger's MAPQ adjustment method[1].

```
 #  Output by aligner:

DP8400008965TLL1C001R0102043364 0   9      133020979      2      100M   *      0      0
    GTTAATGATAACAATGCATCGTAAAACCTTCAGAAGGAAAGGAGAATGTTTTGTGGACCACTTTGGTTTTCTTGTTTGCGTGTGGCAGTTTTAAGTTTTT
    ...
DP8400008965TLL1C001R0102043364 272 7      22510408       2      100M   *      0      0
    AAAAACTTAAAACTGCCACACGCAAACAAGAAAACCAAAGTGGTCCACAAAACATTCTCCTTTCCTTCTGAAGGTTTTACGATGCATTGTTATCATTAAC
    ...
DP8400008965TLL1C001R0102043364 272 6      73517606       2      100M   *      0      0
    AAAAACTTAAAACTGCCACACGCAAACAAGAAAACCAAAGTGGTCCACAAAACATTCTCCTTTCCTTCTGAAGGTTTTACGATGCATTGTTATCATTAAC
    ...

# After adjustment (Seq and Qual in secondary alignments masked as *):

DP8400008965TLL1C001R0102043364 256 9      133020979      0      100M   *      0      0
    *      ...
DP8400008965TLL1C001R0102043364 272 7      22510408       0      100M   *      0      0
    *      ...
DP8400008965TLL1C001R0102043364 16  6      73517606       255    100M   *      0      0
    AAAAACTTAAAACTGCCACACGCAAACAAGAAAACCAAAGTGGTCCACAAAACATTCTCCTTTCCTTCTGAAGGTTTTACGATGCATTGTTATCATTAAC
    ... MM:i:1
```

An example list here to show how to enable mapq adjuestment.

```
PISA sam2bam -@ 2 -report alignment.csv -o out.bam -adjust-mapq -gtf hg38.gtf -qual 255 in.sam
```

2.7 rmdup

To remove the PCA duplication for single-cell experiments, we should consider the cell and molecular barcodes. During the feature counting (processed by `PISA count`), we only rely on the unique UMIs, so it is unnecessary to perform PCR deduplication for the library with UMIs in this application. However, deduplicated BAM is still beneficial for other processes, such as variant calling, peak calling, etc. The following `rmdup` tool is designed to remove duplicated reads which share the same features.

```
# Deduplicate PCR reads with same barcodes.

$ PISA rmdup -tags CB,UR -o rmdup.bam in.bam

Options :
   -tags  [TAGS]      Barcode tags to group reads.
   -@     [INT]       Threads to unpack BAM.
   -o     [BAM]       Output bam.
   -q     [INT]       Map Quality Score cutoff.
   -k                 Keep duplicates, make flag instead of remove them.
   -nw                Disable warnings.

Notice:
* Currently only support single-end reads.
```

In this version, `PISA rmdup` only supports single-end reads. For paired-end reads, such as scATAC data, PCR deduplication can be performed by the `PISA bam2frag` tool.

[1] https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/algorithms/overview#alignment

2.8 pick

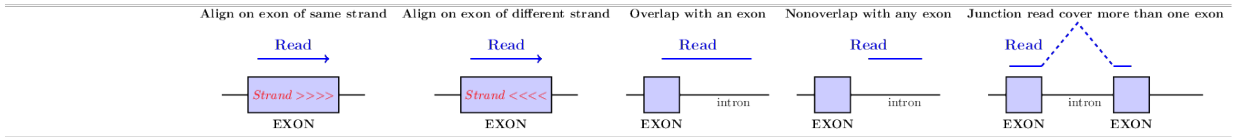
The `PISA pick` tool is designed to select alignments with predefined tags and candidate values.

```
# Pick alignment records with barcode list.

$ PISA pick -tags CB,GN -list cell_barcodes.txt in.bam

Options :
 -tags    [TAGS]      Barcode tags.
 -list    [FILE]      Barcode white list, tag values in related column will be apply.
 -o       [BAM]       Output file.
 -q       [INT]       Map Quality Score cutoff.
 -@       [INT]       Threads to unpack BAM.
```

Depending on how many tags are specified, the candidate list for tags can be one column or more than one column. If more than one tag is set but only one column is in the list, the program will compare the value of the first tag of the alignments with the list. And `pick` tool will only check if other tags exist in the alignments.

2.9 anno

Connect alignment with features is an essential step during single-cell data analysis. We classify various features into three types, gene annotation, functional region, and genetic/sequence variation. For gene annotation, the PISA anno tool first caches all the exons, transcripts, and genes from a GTF database in a tree structure. Then, for each gene, overlapping reads will be compared with each transcript, depending on different alignment states, reads will be grouped into five types, see Fig. 1.



	Align on exon of same strand	Align on exon of different strand	Overlap with an exon	Nonoverlap with any exon	Junction read cover more than one exon
Default parameters	✓	✗	✗	✗	✓
-splice-consider	✓	✗	✓	✗	✓
-ignore-strand	✓	✓	✗	✗	✓
-intron	✓	✗	✓	✓	✓
Region type*	E	A	C	N	S

Figure 1: Gene annotation types and option usage.

```
# annotate strand-specific reads
$ PISA anno -gtf genes.gtf -t 10 -o anno.bam -report annotation_report.csv sorted.bam

# annotate non-strand-specific reads, for Smartseq or bulk RNAseq
$ PISA anno -ignore-strand -gtf genes.gtf -t 10 -o anno.bam -report annotation_report.csv
    sorted.bam

# annotate intron and exon reads, for single-nuclei RNA-seq
$ PISA anno -intron -gtf genes.gtf -t 10 -o anno.bam -report annotation_report.csv sorted.bam
```

For functional region annotation, PISA anno accepts a bed file, which at least has three columns. If no region name is specified in column four, the software will generate an alias name in "chr_start_end" format. Column five of a BED file is quality and will be skipped here. Column six is the strand of the functional region, and strand-sensitive annotation will be enabled if set.

```
$ PISA anno -bed atac_peaks.bed -@ 10 -tag PK -o anno.bam sorted.bam
```

The variant information should be set in a VCF/BCF file for genetic or sequence variant annotation. If the user only has a txt file, the bcftools convert command can convert the flat file to VCF. The software will compare the bases in the alignments and alternative alleles in the VCF and label a new tag for matched records.

```
$ PISA anno -t 10 -vcf variation.vcf.gz in.bam -o anno_vcf.bam
```

Besides these three annotation methods, `PISA anno` also supports a `-chr-species` method. This method requires a binding list for chromosome and related label relationships. The software will check the chromosome and add the related tag for each chromosome. This method, combined with `PISA attrcnt` can help to summarize the mixed two cell lines from different species.

```
# A binding list is tab-separated two columns txt file.
$ cat binding_list.txt
GRCh38_chr1   Human
GRCh38_chr21  Human
mm10_chr21    Mouse

$ PISA anno -t 10 -chr-species binding.txt -btag SP -o anno_species.bam sorted.bam
```

The full options and descriptions list below.

```
# Annotate SAM/BAM records with overlapped function regions. Such as gene, transcript etc.

$ PISA anno -bed peak.bed -tag PK -vcf in.vcf.gz -vtag VF -o anno.bam in.bam
$ PISA anno -gtf genes.gtf -o anno.bam in.bam
$ PISA anno -gtf genes.gtf -o anno.bam -sam in.sam

Options :
 -o       [BAM]      Output bam file.
 -report  [csv]      Summary report.
 -@       [INT]      Threads to compress bam file.
 -q       [0]        Map Quality Score cutoff. MapQ smaller and equal to this value will not be
     annotated.
 -t       [INT]      Threads to annotate.
 -chunk   [INT]      Chunk size per thread.
 -anno-only          Export annotated reads only.
 -sam                Input is SAM file, parse tags from read name.

Options for BED file :
 -bed     [BED]      Function regions. Three or four columns bed file. Col 4 could be empty or
     names of this region.
 -tag     [TAG]      Attribute tag name. Set with -bed.

Options for mixed samples.
 -chr-species [FILE] Chromosome name and related species binding list.
 -btag    [TAG]      Species tag name. Set with -chr-species.

Options for GTF file :
 -gtf     [GTF]      GTF annotation file. gene_id,transcript_id is required for each record.
 -tags    [TAGS]     Attribute names, more details see 'Notice' below. [TX,GN,GX,RE]
 -ignore-strand      Ignore strand of transcript in GTF. Reads mapped to antisense transcripts
     will also be annotated.
 -splice             Reads covered exon-intron edge will also be annotated with all tags.
 -intron             Reads covered intron regions will also be annotated with all tags.
 -tss                Annotate reads start from TSS, designed for capped library. **experiment**
 -ctag    [TAG]      Tag name for TSS annotation. Need set with -tss.
```

```
Options for VCF file :
 -vcf     [VCF/BCF]  Varaints.
 -vtag    [TAG]      Tag name. Set with -vcf.

Notice :
 * If input is SAM format, will try to parse the tags in the read name.
 * For GTF mode, this program will set tags in default, you could also reset them by -tags.
   TX : Transcript id.
   GN : Gene name.
   GX : Gene ID.
   RE : Region type, E (Exon), N (Intron), C (Exon and Intron), S (junction reads cover isoforms
        properly), V (ambiguous reads), I (Intergenic), A (Anitisense)
```

2.10 corr

The diversity of UMIs of each gene in one cell is used to evaluate the gene expression level, but the error of UMI comes from sequencing or PCR may introduce bias. PISA corr was designed to correct the UMI or barcode sequence based on Hamming distance. In default, two groups of UMI from the same gene of one cell with Hamming distance equal to 1, will be considered originating from the same transcript. The group with high frequency will be select as a real one, and another one will be corrected to the high one.

Because `PISA corr` does not require a sorted BAM, this tool will first build a correction list by caching all the raw UMIs and barcodes and then correct them in memory. After these steps, this tool will reread the file and update these records by order. This design can avoid potential bias for the same gene from different chromosomes (i.e., the HLA genes in alternative locus). But this design also required a lot of memory for a big BAM.

CellRanger(2020/07/16) also introduces an algorithm to correct reads with the same UMI of one cell but map to more than one gene[2]. PISA implements this method but not enable it in default. The -cr option can be used by users to enable this function. The reason for not enable it by default is PISA corr not only used to correct UMIs, but also can be used to correct any other types of barcodes. The following example shows how to use PISA corr correct cell barcodes for reads in the same gene.

```
// Reads with same gene tag (GN) and UMI (UB) will be grouped and calculate the Hamming distance
    between each other.
// Only if Hamming distance == 1 will be corrected.
PISA corr -tag CR -new-tag CB -tags-block GN,UB -o cell_barcode_corrected.bam in.bam
```

Full options of `PISA corr` list below.

```
# Correct similar barcodes (hamming distance == 1).

$ PISA corr -tag UR -new-tag UB -tags-block CB,GN -@ 5 -o corr.bam in.bam

Options :
 -o       [BAM]      Output bam.
 -tag     [TAG]      Tag to correct.
 -new-tag [TAG]      Create a new tag for corrected barcodes.
 -tags-block [TAGS]  Tags to define read group. For example, if set to GN (gene), reads in the
     same gene will be grouped together.
 -cr                 Enable CellRanger like UMI correction method. See ‘Examples‘ for details.
 -e                  Maximal hamming distance to define similar barcode, default is 1.
 -@       [INT]      Thread to compress BAM file.


Examples :
```

[2] https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/algorithms/overview

```
// Two groups of reads have same cell barcode (CB) and gene (GN) but their raw UMIs (UR) differ
    by only one base. The UMI of less
// supported is corrected to the UMI with higher support. UB save the checked or corrected UMI.
$ PISA corr -tag UR -new-tag UB -tags-block CB,GN in.bam -o corr.bam

// Same with above. Besides, if two or more groups of reads have same CB and UB but different GN,
    the GN with the most supporting reads
// is kept for UMI counting, and the other read groups are discarded. In case of a tie for
    maximal read support, all read groups are
// discarded, as the gene cannot be confidently assigned (Cell Ranger method).
$ PISA corr -cr -tag UR -new-tag UB -tags-block CB,GN in.bam -o corr.bam
```

2.11 attrcnt

`PISA attrcnt` is used to summarize the meta information of the library. We start introduce this tool with few examples.

```
# Count reads per cell, -cb option is required to specify cell barcode tag
$ PISA attrcnt -cb CB in.bam

// the summary information output in tsv format
BARCODE Raw                     // the title
AGCTATGCTTCTAGTGTAAC-1 38       // cell barcode and raw reads per cell, separated by a tab
GCCGCTGATCGGCCTGCACA-1 12
GTCGCGATTCTGCTCAGAAG-1 13
GTCAGTACCATGCTCAGAAG-1 27
CAACTCGTCGGTTGTCTGAC-1 23

# Count raw reads and reads in the peaks per cell
$ PISA attrcnt -cb CB -tags PK        // PK tag is annotated for reads in peak
            -o summary.tsv          // Summary file
            example/anno/demo_1.bam

$ head summary.tsv
BARCODE Raw    PK
GGCATTATCGGCTCGGTATG  2      2
TGAGTTGTGTATACTCCTAC  2      2
CCGCGGCACTACACACCAGA  1      1
ATTAGTGGTCTCCTGGTCGG  1      1
TTATTGGACCACGTTGAATA  1      1
GGAATGCCACAAGCGCCGTA  1      1
AGTCTATCGTGGCCTGCACA  5      5
AGGCACACCTCGCTGAATTC  3      3
GTCAGGATCGATAACATACG  2      2

# Count UMIs per cell
$ PISA attrcnt -cb CB -tags UB  // UB is tag of corrected UMIs
            -dedup             // -dedup option used to remove duplication of tag values, if not
                set, this
                               // command will export reads with UB tag but not the unique UMIs
                                   per cell
            -o summary.tsv anno.bam

# Count UMIs and Genes per cell
$ PISA attrcnt -cb CB -tags UB,GN // GN is tag for annotated gene; -tags can accept multiple tag
    names, and seperated by ","
            -dedup -o summary.tsv anno.bam
```

```
$ head summary.tsv
BARCODE Raw    UB      GN
AGCTATGCTTCTAGTGTAAC-1 38    35      0
GCCGCTGATCGGCCTGCACA-1 12    12      0
GTCGCGATTCTGCTCAGAAG-1 13    11      0
GTCAGTACCATGCTCAGAAG-1 27    26      0
CAACTCGTCGGTTGTCTGAC-1 23    21      0
GGTACACCACAGTAGTTACG-1 12    10      0
GCGCGCCGAGGGACACTCTT-1 1     1       0
CTCTAAGCATCGAGGTTAAC-1 162   138     50
TTCGTAGCACCGATACTAGC-1 51    48      46
```

Full list of options list below.

```
# Count the frequency of tag values.

$ PISA attrcnt -cb CB -tags UR,GN -dedup -all-tags in.bam

Options :
 -cb       [TAG]     Cell Barcode, or other tag used for grouping reads.
 -list     [FILE]    Cell barcode white list.
 -tags     [TAGS]    Tags to count.
 -dedup              Deduplicate the atrributes in each tag.
 -all-tags           Only records with all tags be count.
 -group    [TAG]     Group tag, count all tags for each group seperately.
 -o        [FILE]    Output count table.
 -q        [INT]     Map Quality to filter bam.
 -no-header          Ignore header in the output.
 -@        [INT]     Thread to unpack bam.
 -ttag     [TAG]     Region type tag. [RE]
 -ttype              Region type used to count. Set ‘E,S‘ to count exon enclosed reads. Set ‘N,C‘
     to count intron overlapped reads.
```

2.12 extract

`PISA extract` is designed to extract the values of tags from BAM records and generate a tab-separated file.

```
# Extract tag values from alignments.

$ PISA extract -tags CB,UR,GN -o tags.tsv in.bam

Options :
 -tags     [TAGS]    Tags to be extracted.
 -o        [FILE]    Output file. tsv format
 -n                  Print read name.
 -q                  Map Quality Score threshold.
 -all                Only export if all tags have value.
```

2.13 count

The `PISA count` is designed to generate a counts matrix for various features/tags. It generates a gene x cell digit matrix before v0.4. From v0.4, this tool supports the MEX format output. And it's highly recommended to use the MEX file in downstream analysis for better performance. Remember to use `-outdir` to specify the output files; consider MEX format consists of three files. The following example was published on the wiki page online, and I made some minor changes and put it here for quick reading.

```
# Gene expression
## Count gene expression from one bam file
```

```
# The following command will count the gene expression of all cells (raw) in the bam file.

$ PISA count -tag CB    // cellular identify tag
          -anno-tag GN // gene tag, only reads with this annotation will be count
          -outdir raw_cell_gene_expression // Gene X Cell matrix in MEX format
          -q 20       // only count reads with mapping quality >= 20
          anno.bam    // annotated bam file

# Count gene expression from multiple bam files.
# This is useful not only when you have hundreds of bams and one bam contains one cell but also
    works for one library sequenced several times.

$ PISA count -file-barcode         // use alias name for each bam file as cell barcode. If this
    flag is not set -cb must be specified.
          -sample-list bam_list.txt // bam file path and alias name
          -outdir exp/ -@ 5 -anno-tag GN


# The '-sample-list' accepts a tab-separated text file. The first column is the bam file path, and
    the second column is the optional alias cell name. If '-file-barcode' is set, the second
    column will be used as cell barcodes. A demo list below.

$ head bam_list.txt
/home/shiquan/Downloads/velocity/anno/A10.bam A10
/home/shiquan/Downloads/velocity/anno/A11.bam A11
/home/shiquan/Downloads/velocity/anno/A12.bam A12
/home/shiquan/Downloads/velocity/anno/A13.bam A13
/home/shiquan/Downloads/velocity/anno/A14.bam A14
/home/shiquan/Downloads/velocity/anno/A15.bam A15
/home/shiquan/Downloads/velocity/anno/A16.bam A16
/home/shiquan/Downloads/velocity/anno/A17.bam A17
/home/shiquan/Downloads/velocity/anno/A18.bam A18
/home/shiquan/Downloads/velocity/anno/A19.bam A19



## Count gene expression from scRNA with UMIs
Gene expression of each cell usually can be estimated by counting unique UMIs.

$ PISA count -tag CB
          -umi UB     // UMI tag, for each gene only count unique UMI instead of aligned reads
              number
          -anno-tag GN -outdir raw_cell_gene_expression -q 20 anno.bam

## Count gene expression with known cell barcodes
# The following command will export gene expression counts of cells in the cell barcode list only.

$ PISA count -tag CB
          -list cell_barcodes.txt // cell barcodes whitelist
          -umi UB -anno-tag GN -outdir filtered_cell_gene_expression -q 20 anno.bam

## Count spliced and unspliced reads for RNA velocity analysis, this step requires reannotate
    intron reads in the BAM

$ PISA anno -gtf genes.gtf -intron -o anno.bam in.bam // make sure -intron option used in
    annotation step, otherwise GN tag will not be labeled for intronic reads

$ PISA count -tag CB -list cell_barcodes.txt
```

```
          -velo     // enable velocity analysis mode
          -umi UB -anno-tag GN -outdir exp/ anno.bam


# Peak signals
# Reads overlapped with each peak or labeled by any other tags can be counted with the following
    method.

$ PISA count -tag CB                  // cellular identify tag
          -anno-tag PK               // Peak tag
          -outdir raw_cell_peak_signal // Peak X Cell signal file in MEX format
          anno.bam
```

2.13.1 Description of MEX file

The Market Exchange (MEX) format[3] is designed for representing the sparse matrix. The `-outdir` option specify the output directory for one MEX fold. The MEX format consists of three files, one is cell barcodes, one is feature names (genes or peak names), and the third one defines the expression or signal values.

```
$ ls
barcodes.tsv.gz features.tsv.gz matrix.mtx.gz

$ zcat barcodes.tsv.gz|head
AACCTGGTGAAGTTGTCGAA
AAGGAACTAAGCGCAGCACC
CGATAGAATACTTCTTCGTA
TACTATCCTCTAGCTGCTAC
TGACCATCCTACAGTCCACC
CAGATTCAACTACGAAGTGC
TTCGTAGCACTCTTCATCTC
GGCACCTTGCTTAACGTAGG
ACTTCGGATACGTATCGCCT
GACTCGCTAGTAGTCGGAAT

$ zcat features.tsv.gz|head
RP11-34P13.7
RP11-34P13.8
RP11-34P13.9
FO538757.3
FO538757.2
AP006222.2
RP4-669L17.10
RP5-857K21.4
RP11-206L10.4
RP11-206L10.9

$ zcat matrix.mtx.gz|head
%%MatrixMarket matrix coordinate integer general
% Generated by PISA v0.4-alpha-72-g09c4ded
23900  782761 11533380
1      1      2
1      2      2
1      3      2
1      4      1
1      5      2
1      6      1
```

[3] https://math.nist.gov/MatrixMarket/formats.html

1 7 2

Here I generate R and Python API to load the MEX files. The following code can be also found at PISA/R directory.

```
### *R*

#' Read feature count matrix generated by ‘PISA count‘.
#'
#' This function will read Matrix Market files from a directory which generated by ‘PISA count‘.
#' @import Matrix
#' @param mex_dir Feature count outdir generated by ‘PISA count‘.
#' @return Returns a sparse matrix of feature counts or a list of spliced, unspliced, and
#'         spanning reads sparse matrix.
#'
#' @export
ReadPISA <- function(mex_dir=NULL,
                     barcode.path = NULL,
                     feature.path = NULL,
                     matrix.path=NULL,
                     use_10X=FALSE) {
  if (is.null(mex_dir) && is.null(barcode.path) && is.null(feature.path) &&
       is.null(matrix.path)) {
    stop("No matrix set.")
  }
  if (!is.null(mex_dir) && !file.exists(mex_dir) ) {
    stop(paste0(mex_dir, " does not exist."))
  }
  if (is.null(barcode.path) && is.null(feature.path) && is.null(matrix.path)) {
    barcode.path <- paste0(mex_dir, "/barcodes.tsv.gz")
    feature.path <- paste0(mex_dir, "/features.tsv.gz")
    matrix.path <- paste0(mex_dir, "/matrix.mtx.gz")
  }
  spliced.path <- paste0(mex_dir, "/spliced.mtx.gz")
  unspliced.path <- paste0(mex_dir, "/unspliced.mtx.gz")
  spanning.path <- paste0(mex_dir, "/spanning.mtx.gz")

  if (!file.exists(barcode.path) || !file.exists(feature.path)) {
    stop(paste0("No expression file found at ", mex_dir))
  }

  .ReadPISA0 <- function(barcode.path, feature.path, matrix.path, use_10X) {
    mat <- Matrix::readMM(file = matrix.path)
    feature.names <- read.delim(feature.path,
                                header = FALSE,
                                stringsAsFactors = FALSE
                                )
    barcode.names <- read.delim(barcode.path,
                                header = FALSE,
                                stringsAsFactors = FALSE
                                )
    colnames(mat) <- barcode.names$V1
    if (use_10X == TRUE) {
      rownames(mat) <- make.unique(feature.names$V2)
    } else {
      rownames(mat) <- make.unique(feature.names$V1)
    }
    mat
  }
```

```
  if (!file.exists(spliced.path) && file.exists(matrix.path)) {
    return(.ReadPISA0(barcode.path, feature.path, matrix.path, use_10X))
  }
  mat <- list()
  cat("Load spliced matrix ...\n")
  mat$spliced <- Matrix::readMM(file = spliced.path)
  cat("Load unspliced matrix ...\n")
  mat$unspliced <- Matrix::readMM(file = unspliced.path)
  cat("Load spanning matrix ...\n")
  mat$spanning <- Matrix::readMM(file = spanning.path)

  feature.names <- read.delim(feature.path,
                              header = FALSE,
                              stringsAsFactors = FALSE
  )
  barcode.names <- read.delim(barcode.path,
                              header = FALSE,
                              stringsAsFactors = FALSE
  )
  colnames(mat$spliced) <- barcode.names$V1
  rownames(mat$spliced) <- make.unique(feature.names$V1)
  colnames(mat$unspliced) <- barcode.names$V1
  rownames(mat$unspliced) <- make.unique(feature.names$V1)
  colnames(mat$spanning) <- barcode.names$V1
  rownames(mat$spanning) <- make.unique(feature.names$V1)
  mat
}
```

And for Python.

```
import pandas as pd
import scipy.io
import anndata
from scipy.sparse import csr_matrix
def ReadPISA(path):
    mat = scipy.io.mmread(path+"/"+"matrix.mtx.gz").astype("float32")
    mat = mat.transpose()
    mat = csr_matrix(mat)
    adata = anndata.AnnData(mat,dtype="float32")
    genes = pd.read_csv(path+'/'+'features.tsv.gz', header=None, sep='\t')
    var_names = genes[0].values
    var_names = anndata.utils.make_index_unique(pd.Index(var_names))
    adata.var_names = var_names
    adata.var['gene_symbols'] = genes[0].values
    adata.obs_names = pd.read_csv(path+'/'+'barcodes.tsv.gz', header=None)[0].values
    adata.var_names_make_unique()
    return adata
```

Options and descriptions list below.

```
# Count reads or fragments matrix for single-cell datasets.

$ PISA count -cb CB -anno-tag GN -umi UB -outdir exp aln.bam
$ PISA count [options] aln1.bam,aln2.bam
$ PISA count -file-barcode -sample-list bam_files.txt -outdir exp

Options :
 -cb       [TAG]     Cell barcode tag.
 -anno-tag [TAG]     Annotation tag, gene or peak.
```

```
 -list    [FILE]    Barcode white list, used as column names at matrix. If not set, all barcodes
     will be count.
 -outdir  [DIR]     Output matrix in MEX format into this fold.
 -umi     [TAG]     UMI tag. Count once if more than one record has same UMI in one gene or peak.
 -one-hit           Skip if a read hits more than 1 gene or peak.
 -q       [INT]     Minimal map quality to filter. Default is 20.
 -@       [INT]     Threads to unpack BAM.
 -ttag    [TAG]     Region type tag. [RE]
 -velo              Generate spliced and unspliced matrix files for RNA velocity analysis.
 -ttype   [TYPE]    Region type used to count. Set 'E,S' to count exon enclosed reads. Set 'N,C'
     to count intron overlapped reads.
 -file-barcode      No cell barcode tag in the bam, but alias file name as cell barcode. This
     option must use with -sample-list.
 -sample-list [FILE] A list of bam files. First column of this file should be path of bam files.
     Optional second column is the
                    sample or cell name. This option is useful for one cell per bam experiment,
                        like Smartseq.

Notice :
 * Region type (RE), which label functional region reads mapped, is annotated by 'PISA anno'.
     Optional -ttype can be set
   to one of region types(E/S/C/N) or combination to count reads mapped to these functional
       regions only.
 * If you want count from more than one bam file, there are two ways to set the parameter. By
     seperating bam files with ',' or by
   setting -sample-list option. But if you want alias each bam with a predefined cell name, only
       -sample-list supported.
 * -cb conflict with -file-barcode. PISA read cell barcode from bam tag or alias name list. Not
     both.
 * If -velo set, spliced and unspliced folders will be created at outdir.
```

2.14 bam2fq

`PISA bam2fq` is designed to convert alignment records to FATSQ+ records. Option `-tags` specify which tags will be kept in the FASTQ+. Full options list below.

```
# Convert BAM into fastq.
$ PISA bam2fq -tags CB,UB,GN -o out.fq aln.bam

Options :
 -tags    [TAGS]    Export tags in read name.
 -f                 Filter records if specified tags not all exist.
 -fa                Output fasta instead of fastq.
 -o       [fastq]   Output file.
 -@       [INT]     Threads to unpack BAM.
```

2.15 bam2frag

The fragment file is a five columns tab-separated flat file. The first column is the chromosome name, and the second column is the start location of this fragment (0 based), and the third column is the end position in 1 based of this fragment. The fourth column is the cell barcode of this fragment and the last column is how many duplicates of this fragment.

`PISA bam2frag` requires the input BAM file to be sorted by coordinate. And this tool will check the cell barcode and the fragment position for each paired reads, duplicates in one cell will only keep one record, and the how many copies for this fragment will be updated in column four.

```
# Convert sam record to fragment file.
```

```
$ PISA bam2frag -cb CB -list cell_barcodes.txt -o out.tsv.gz in.bam

Options:
 -o       [FILE]   Output file. This file will be bgzipped and indexed.
 -cb      [TAG]    Cell barcode tag.
 -list    [FILE]   Cell barcode white list.
 -q       [20]     Mapping quality score to filter reads.
 -isize   [2000]   Skip if insert size greater than this. [2KB]
 -bed     [BED]    Only convert fragments overlapped with target regions.
 -black-region [BED] Skip convert fragments overlapped with black regions.
 -stat    [FILE]   Transposition events per cell.
 -@       [4]      Thread to unpack and pack files.[4]
 -disable-offset  Disable Tn5 offset for each fragment.
```

2.16 depth

`PISA depth` generates coverage information for each position of the predefined region(s). The significant difference between `PISA depth` and `samtools depth` [4] is that PISA is strand sensitive, and PISA can produce results for target cells.

```
# Count coverage depth or unique UMIs for genome locations.

Usage : PISA depth [options] sorted.bam [region]

$ PISA depth -cb CB -umi UB -tags GN -region in.bed -o depth.tsv sorted.bam
$ PISA depth -cb CB -umi UB sorted.bam chr1:1-2:+

Options :
 -tag     [TAG]     Tag used for grouping reads.
 -list    [FILE]    Candidate list for -tag.
 -umi     [TAG]     UMI tag. If set, only count unique UMIs for each location.
 -bed     [BED]     Target BED region file. If the strand in column six set, only count reads with
     the same strand.
 -o       [FILE]    Output depth file. [stdout].
 -q       [INT]     Minimal map quality to filter. [20]
 -@       [INT]     Threads to unpack bam. [4]

Notice :
 * Require sorted and indexed BAM as input.
 * Compare with 'samtools depth', PISA depth considers UMIs and strand of reads.
```

2.17 count2

`PISA count` generates the MEX format matrix for fragments per peak.

```
# Count fragments per peak per cell matrix.

$ PISA count2 -bed peaks.bed -t 10 -list barcodes.txt -outdir exp fragments.tsv.gz

Options :
 -list    [FILE]    Barcode white list, used as column names at matrix. If not set, all barcodes
     will be count.
 -outdir  [DIR]     Output matrix in MEX format into this fold.
 -prefix  [STR]     Prefix of output files.
 -t       [INT]     Threads.
```

[4] http://www.htslib.org/doc/samtools-depth.html

2.18 Other experimental tools.

Tools not listed above are still under development and may contain bugs or mistakes.