## Contents

1 **Installation**  
   1.1 Dependencies ................................................. 3  
   1.2 Installing the latest code from github (recommended) .................................................. 3  
   1.3 Installing a particular release ......................................... 3  
   1.4 To run using docker ............................................ 4  

2 **Quickstart - how to polish a genome assembly**  
   2.1 Download example dataset ......................................... 5  
   2.2 Analysis workflow .............................................. 6  
   2.3 Data preprocessing ............................................... 6  
   2.4 Compute the draft genome assembly using canu  .................................................. 7  
   2.5 Compute a new consensus sequence for a draft assembly ............................................. 7  
   2.6 Evaluate the assembly ........................................... 8  

3 **Quickstart - how to align events to a reference genome**  
   3.1 Download example dataset ......................................... 9  
   3.2 Align the reads with minimap2 ...................................... 10  
   3.3 Align the nanopore events to a reference  .................................................. 10  
   3.4 Assess the eventalign output ........................................ 10  
   3.5 Example plots .................................................. 11  

4 **Quickstart - calling methylation with nanopolish**  
   4.1 Download example dataset ......................................... 13  
   4.2 Data preprocessing ............................................... 14  
   4.3 Aligning reads to the reference genome .................................................. 14  
   4.4 Calling methylation ................................................ 14  

5 **Quickstart - how to estimate poly(A) tail lengths from nanopore native RNA reads**  
   5.1 Software requirements ............................................. 17  
   5.2 Download raw fast5 data and basecall  .................................................. 17  
   5.3 Index with nanopolish index .......................................... 18  
   5.4 Align with minimap2 and format the BAM file .................................................. 18  
   5.5 Segmentation and tail length estimation with nanopolish polya ...................................... 19  
   5.6 Interpreting the output TSV file ........................................ 19  

6 **Helping us debug nanopolish**  
   6.1 Overview .................................................. 21
nanopolish is a software package for signal-level analysis of Oxford Nanopore sequencing data. Nanopolish can calculate an improved consensus sequence for a draft genome assembly, detect base modifications, call SNPs and indels with respect to a reference genome and more (see Nanopolish modules, below).
CHAPTER 1

Installation

1.1 Dependencies

A compiler that supports C++11 is needed to build nanopolish. Development of the code is performed using gcc-4.8. By default, nanopolish will download and compile all of its required dependencies. Some users however may want to use system-wide versions of the libraries. To turn off the automatic installation of dependencies set HDF5=noinstall, EIGEN=noinstall or HTS=noinstall parameters when running make as appropriate. The current versions and compile options for the dependencies are:

- libhdf5-1.8.14 compiled with multi-threading support --enable-threadsafe
- eigen-3.2.5
- htslib-1.4

Additionally the helper scripts require biopython and pysam.

1.2 Installing the latest code from github (recommended)

You can download and compile the latest code from github as follows

```
git clone --recursive https://github.com/jts/nanopolish.git
cd nanopolish
make
```

1.3 Installing a particular release

When major features have been added or bugs fixed, we will tag and release a new version of nanopolish. If you wish to use a particular version, you can checkout the tagged version before compiling
1.4 To run using docker

First build the image from the dockerfile:

docker build .

Note the uuid given upon successful build. Then you can run nanopolish from the image:

docker run -v /path/to/local/data/data:/data -it :image_id ./nanopolish eventalign –r /data/reads.fa –b /data/alignments.sorted.bam –g /data/ref.fa
Quickstart - how to polish a genome assembly

The original purpose of nanopolish was to improve the consensus accuracy of an assembly of Oxford Nanopore Technology sequencing reads. Here we provide a step-by-step tutorial to help you get started.

Requirements:

- nanopolish
- samtools
- minimap2
- MUMmer

2.1 Download example dataset

You can download the example dataset we will use here:

wget http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz
tar -xvf ecoli_2kb_region.tar.gz
cd ecoli_2kb_region

Details:

- Sample: E. coli str. K-12 substr. MG1655
- Instrument: MinION sequencing R9.4 chemistry
- Basecaller: Albacore v2.0.1
- Region: “tig00000001:200000-202000”
- Note: Ligation-mediated PCR amplification performed

This is a subset of reads that aligned to a 2kb region in the E. coli draft assembly. To see how we generated these files please refer to the tutorial creating_example_dataset.

You should find the following files:
• **reads.fasta**: subset of basecalled reads
• **draft.fa**: draft genome assembly
• **draft.fa.fai**: draft genome assembly index
• **fast5_files/**: a directory containing FAST5 files
• **ecoli_2kb_region.log**: a log file for how the dataset was created with nanopolish helper script (scripts/extract_reads_aligned_to_region.py)

For the evaluation step you will need the reference genome:

```
```

## 2.2 Analysis workflow

The pipeline below describes the recommended analysis workflow for larger datasets. In this tutorial, we will run through the basic steps of the pipeline for this smaller (2kb) dataset.

### 2.3 Data preprocessing

nanopolish needs access to the signal-level data measured by the nanopore sequencer. To begin, we need to create an index `readdb` file that links read ids with their signal-level data in the FAST5 files:
nanopolish index -d fast5_files/ reads.fasta

We get the following files: reads.fasta.index, reads.fasta.index.fai, reads.fasta.index.gz, and reads.fasta.index.readdb.

2.4 Compute the draft genome assembly using canu

As computing the draft genome assembly takes a few hours we have included the pre-assembled data for you (draft.fa). We used the following parameters with canu:

```bash
canu 
  -p ecoli  
  -d outdir genomeSize=4.6m 
  -nanopore-raw albacore-2.0.1-merged.fastq
```

2.5 Compute a new consensus sequence for a draft assembly

Now that we have reads.fasta indexed with nanopolish index, and have a draft genome assembly draft.fa, we can begin to improve the assembly with nanopolish. Let us get started!

First, we align the original reads (reads.fasta) to the draft assembly (draft.fa) and sort alignments:

```bash
minimap2 -ax map-ont -t 8 draft.fa reads.fasta | samtools sort -o reads.sorted.bam -T reads.tmp
samtools index reads.sorted.bam
```

**Checkpoint:** we can do a quick check to see if this step worked. The bam file should not be empty.

```bash
samtools view reads.sorted.bam | head
```

Then we run the consensus algorithm. For larger datasets we use nanopolish_merangerange.py to split the draft genome assembly into 50kb segments, so that we can run the consensus algorithm on each segment in parallel. The output would be the polished segments in fasta format. Since our dataset is only covering a 2kb region, we skip this step and use the following command:

```bash
nanopolish variants --consensus -o polished.vcf 
  -w "tig00000001:200000-202000" 
  -r reads.fasta 
  -b reads.sorted.bam 
  -g draft.fa
```

We are left with: polished.vcf.

**Note:** As of v0.10.1, nanopolish variants --consensus only outputs a VCF file instead of a fasta sequence.

To generate the polished genome in fasta format:

```bash
nanopolish vcf2fasta --skip-checks -g draft.fa polished.vcf > polished_genome.fa
```

We only polished a 2kb region, so let’s pull that out:

```bash
samtools faidx polished_genome.fa
samtools faidx polished_genome.fa "tig00000001:200000-202000" > polished.fa
```
2.6 Evaluate the assembly

To analyze how nanopolish performed improving the accuracy we use MUMmer. MUMmer contains “dnadiff”, a program that enables us to see a report on alignment statistics. With dnadiff we can compare the two different assemblies.

```bash
mkdir analysis
MUMmer3.23/dnadiff --prefix analysis/draft.dnadiff ref.fa draft.fa
MUMmer3.23/dnadiff --prefix analysis/polished.dnadiff ref.fa polished.fa
```

This generates `draft.dnadiff.report` and `polished.dnadiff.report` along with other files. The metric we are interested in is AvgIdentity under [ Alignments ] 1-to-1, which is a measurement of how similar the genome assemblies are to the reference genome. We expect to see a higher value for the polished assembly than the draft (99.90 vs 99.53), concluding that the nanopolish consensus algorithm worked successfully.

**Note:** The example dataset was PCR amplified causing a loss of methylation information. We recommend using the `-q dam,dcm` with nanopolish `variants --consensus` if you have data with methylation information to account for known bacterial methyltransferases.
CHAPTER 3

Quickstart - how to align events to a reference genome

The eventalign module in nanopl指挥 is used to align events or “squiggles” to a reference genome. We (the developers of nanopolish) use this feature extensively when we want to see what the low-level signal information looks like. It helps us model the signal and discover differences in current that might hint at base modifications. Here we provide a step-by-step tutorial to help you get started with the nanopl指挥 eventalign module.

For more information about eventalign:

- Blog post: “Aligning Nanopore Events to a Reference”
- Paper: “A complete bacterial genome assembled de novo using only nanopore sequencing data”

Requirements:

- nanopolish
- samtools
- minimap2

3.1 Download example dataset

You can download the example dataset we will use here:

```bash
wget http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz
tar -xvf ecoli_2kb_region.tar.gz
cd ecoli_2kb_region
```

Details:

- Sample: E. coli str. K-12 substr. MG1655
- Instrument: MinION sequencing R9.4 chemistry
- Basecaller: Albacore v2.0.1
- Region: “tig00000001:200000-202000”
Nanopolish Documentation, Release 0.8.4

- Note: Ligation-mediated PCR amplification performed

This is a subset of reads that aligned to a 2kb region in the E. coli draft assembly. To see how we generated these files please refer to this section: Tutorial - using extraction helper script to create example datasets.

You should find the following files:
- reads.fasta: subset of basecalled reads
- fast5_files/: a directory containing FAST5 files

You will need the E. coli reference genome:


3.2 Align the reads with minimap2

We first need to index the reads:
nanopolish index -d fast5_files/ reads.fasta

Output files: reads.fasta.index, reads.fasta.index.fai, reads.fasta.index.gzi, and reads.fasta.index.readdb.

Then we can align the reads to the reference:

minimap2 -ax map-ont -t 8 ref.fa reads.fasta | samtools sort -o reads-ref.sorted.bam -T reads.tmp
samtools index reads-ref.sorted.bam

Output files: reads-ref.sorted.bam and reads-ref.sorted.bam.bai.

Checkpoint: Let’s see if the bam file is not truncated. This will check that the beginning of the file contains a valid header, and checks if the EOF is present. This will exit with a non-zero exit code if the conditions were not met:
samtools quickcheck reads-ref.sorted.bam

3.3 Align the nanopore events to a reference

Now we are ready to run nanopolish to align the events to the reference genome:
nanopolish eventalign
--reads reads.fasta 
--bam reads-ref.sorted.bam 
--genome ref.fa 
--scale-events > reads-ref.eventalign.txt

3.4 Assess the eventalign output

If we take a peek at the first few lines of reads-ref.eventalign.txt this is what we get:
### 3.5 Example plots

In Figure 1 of our methylation detection paper we show a histogram of `event_level_mean` for a selection of k-mers to demonstrate how methylation changes the observed current. The data for these figures was generated by eventalign, which we subsequently plotted in R using ggplot2.
Quickstart - calling methylation with nanopolish

Oxford Nanopore sequencers are sensitive to base modifications. Here we provide a step-by-step tutorial to help you get started with detecting base modifications using nanopolish.

For more information about our approach:


Requirements:
- nanopolish v0.8.4
- samtools v1.2
- minimap2

4.1 Download example dataset

In this tutorial we will use a subset of the NA12878 WGS Consortium data. You can download the example dataset we will use here (warning: the file is about 2GB):

```bash
wget http://s3.climb.ac.uk/nanopolish_tutorial/methylation_example.tar.gz
tar -xvf methylation_example.tar.gz
cd methylation_example
```

Details:
- Sample : Human cell line (NA12878)
- Basecaller : Guppy v2.3.5
- Region: chr20:5,000,000-10,000,000

In the extracted example data you should find the following files:
- `albacore_output.fastq`: the subset of the basecalled reads
Nanopolish Documentation, Release 0.8.4

- **reference.fasta**: the chromosome 20 reference sequence
- **fast5_files/**: a directory containing signal-level FAST5 files

The reads were basecalled using this albacore command:

```
guppy_basecaller -c dna_r9.4.1_450bps_flipflop.cfg -i fast5_files/ -s basecalled/
```

After the basecaller finished, we merged all of the fastq files together into a single file:

```
cat basecalled/*.fastq > output.fastq
```

### 4.2 Data preprocessing

nanopolish needs access to the signal-level data measured by the nanopore sequencer. To begin, we need to create an index file that links read ids with their signal-level data in the FAST5 files:

```
nanopolish index -d fast5_files/ output.fastq
```

We get the following files: `output.fastq.index`, `output.fastq.index.fai`, `output.fastq.fastq.index.gzi`, and `output.fastq.index.readdb`.

### 4.3 Aligning reads to the reference genome

Next, we need to align the basecalled reads to the reference genome. We use minimap2 as it is fast enough to map reads to the human genome. In this example we’ll pipe the output directly into samtools sort to get a sorted bam file:

```
minimap2 -a -x map-ont reference.fasta output.fastq | samtools sort -T tmp -o output.sorted.bam
samtools index output.sorted.bam
```

### 4.4 Calling methylation

Now we’re ready to use nanopolish to detect methylated bases (in this case 5-methylcytosine in a CpG context). The command is fairly straightforward - we have to tell it what reads to use (`output.fastq`), where the alignments are (`output.sorted.bam`), the reference genome (`reference.fasta`) and what region of the genome we’re interested in (chr20:5,000,000-10,000,000):

```
nanopolish call-methylation -t 8 -r output.fastq -b output.sorted.bam -g reference.fasta -w "chr20:5,000,000-10,000,000" > methylation_calls.tsv
```

The output file contains a lot of information including the position of the CG dinucleotide on the reference genome, the ID of the read that was used to make the call, and the log-likelihood ratio calculated by our model:

<table>
<thead>
<tr>
<th>chromosome</th>
<th>start</th>
<th>end</th>
<th>read_name</th>
<th>log_lik_ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr20</td>
<td>4980553</td>
<td>4980553</td>
<td>cle202f4-e8f9-4eb8-b9a6-d79e6fable9a</td>
<td>3.70</td>
</tr>
<tr>
<td></td>
<td>-167.47</td>
<td>-171.17</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>chr20</td>
<td>4980599</td>
<td>4980599</td>
<td>cle202f4-e8f9-4eb8-b9a6-d79e6fable9a</td>
<td>2.64</td>
</tr>
<tr>
<td></td>
<td>-98.87</td>
<td>-101.51</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
A positive value in the `logLikRatio` column indicates support for methylation. We have provided a helper script that can be used to calculate how often each reference position was methylated:

```
scripts/calculate_methylation_frequency.py methylation_calls.tsv > methylation_frequency.tsv
```

The output is another tab-separated file, this time summarized by genomic position:

<table>
<thead>
<tr>
<th>chromosome</th>
<th>start</th>
<th>end</th>
<th>num_cpgs_in_group</th>
<th>called_sites</th>
<th>called_sites_methylated</th>
<th>frequency_group_sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr20</td>
<td>5036763</td>
<td>5036763</td>
<td>1</td>
<td>21</td>
<td>20</td>
<td>GCCACCGCGCCCA</td>
</tr>
<tr>
<td></td>
<td>0.952</td>
<td></td>
<td>split-group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chr20</td>
<td>5036770</td>
<td>5036770</td>
<td>1</td>
<td>21</td>
<td>20</td>
<td>GTATACCGCGTCC</td>
</tr>
<tr>
<td></td>
<td>0.964</td>
<td></td>
<td>split-group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chr20</td>
<td>5036780</td>
<td>5036780</td>
<td>1</td>
<td>21</td>
<td>20</td>
<td>AAGGACGTTAT</td>
</tr>
<tr>
<td></td>
<td>0.952</td>
<td></td>
<td>split-group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chr20</td>
<td>5037173</td>
<td>5037173</td>
<td>1</td>
<td>13</td>
<td>5</td>
<td>AGTCCCGATAT</td>
</tr>
<tr>
<td></td>
<td>0.385</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the example data set we have also included bisulfite data from ENCODE for the same region of chromosome 20. We can use the included `compare_methylation.py` helper script to do a quick comparison between the nanopolish methylation output and bisulfite:

```
python3 compare_methylation.py bisulfite.ENCFF835NTC.example.tsv methylation_frequency.tsv > bisulfite_vs_nanopolish.tsv
```

We can use R to visualize the results - we observe good correlation between the nanopolish methylation calls and bisulfite:

```r
library(ggplot2)
library(RColorBrewer)
data <- read.table("bisulfite_vs_nanopolish.tsv", header=T)

# Set color palette for 2D heatmap
rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))
r <- rf(32)
c <- cor(data$frequency_1, data$frequency_2)
title <- sprintf("N = %d r = %.3f", nrow(data), c)
```
Here’s what the output should look like:
Quickstart - how to estimate poly(A) tail lengths from nanopore native RNA reads

Owing to homopolymer effects and the proximity to the sequencing adapters, the polyadenylated tails of reads obtained from nanopore native RNA sequencing are improperly basecalled, making their lengths difficult to quantify. We developed the *polya* subprogram to use an alternative statistical model to estimate these tail lengths.

In this quickstart tutorial, we’ll show you how to estimate polyadenylated tail lengths step-by-step, starting from nothing but raw fast5 files. We’ll basecall the fast5 files with Oxford Nanopore Technologies’ *albacore* basecaller, before aligning the resulting reads with *minimap2*, indexing the files with *nanopolish index*, and finally segmenting the reads and calling the tail lengths with *nanopolish polya*.

We’ll be following the steps taken in our benchmarking analysis workflow that accompanies our publication. In each step below, we’ll generate files in the working directory instead of making subdirectories, except in the case of large datasets.

### 5.1 Software requirements

- *nanopolish* >= v10.2
- *minimap2* >= v2.12
- *samtools* >= v1.9
- *albacore* >= v2.3.3 (from Oxford Nanopore’s private customer portal)

### 5.2 Download raw fast5 data and basecall

Let’s start by downloading a dataset of fast5 files from the European Nucleotide Archive. We’ll download a tarball of fast5 files containing reads that are known to have polyadenylated tail lengths of 30nt.
mkdir data && mkdir data/fastqs
wget ftp://ftp.sra.ebi.ac.uk/vol1/run/ERR276/ERR2764784/30xpolyA.tar.gz -O 30xpolyA.tar.gz && mv 30xpolyA.tar.gz data/
tar -xzf data/30xpolyA.tar.gz -C data/
guppy_basecaller -c rna_r9.4.1_70bps_hac.cfg -i data/30xpolyA/fast5/ -s data/
cat data/fastqs/*.fastq > data/30xpolyA.fastq

In the above, change the value of the -f and -k arguments based on your flow-cell and sequencing kit, as the basecaller’s accuracy is highly dependent upon these settings.

Our directory structure should now look something like this:

```
(current directory)
|-- data/
 | -- 30xpolyA.fastq
 | -- 30xpolyA.tar.gz
 | -- 30xpolyA/
 |   -- fast5/
 |     -- pass/
 |     ----- raw_read1.fast5
 |     ----- (... more raw fast5's here ...)
 | -- fastqs/
 | -- workspace/
 | -- pass/
```

### 5.3 Index with nanopolish index

Let’s construct an index for our reads with nanopolish’s `index` subprogram. This constructs a fast lookup table that tells our program where to find the raw fast5 file for each basecalled read.

```
nanopolish index --directory=data/30xpolyA/fast5/ pass --sequencing-summary=data/fastqs/sequencing_summary.txt data/30xpolyA.fastq
```

This should generate a collection of files in the same directory that contains the `30xpolyA.fastq`.

### 5.4 Align with minimap2 and format the BAM file

Now we’ll align our basecalled mRNA sequences in the fastq file with our reference. First download a reference fasta:

```
mv enolase_reference.* data/
```

Note that our directory structure should now look like this:

```
(current directory)
|-- data/
 | -- enolase_reference.fas
 | -- enolase_reference.fai
 | -- (... same structure as above ...)
```

18  Chapter 5. Quickstart - how to estimate poly(A) tail lengths from nanopore native RNA reads
Let’s run `minimap2` to align our basecalled reads to this reference and generate a collection of SAM/BAM files with `samtools`:

```
minimap2 -a -x map-ont data/enolase_reference.fas data/30xpolyA.fastq | samtools view -b - -o data/30xpolyA.bam
cd data && samtools sort -T tmp -o 30xpolyA.sorted.bam 30xpolyA.bam && samtools index 30xpolyA.sorted.bam && cd ..
```

Note that we used `-x map-ont`, which is typically for unspliced reads (e.g. genomic DNA) coming from nanopore sequencers. In typical native mRNA sequencing, you should use `-x splice`, which is a splice-aware setting (and uses a different gap cost in the alignment). We’re using `map-ont` due to the fact that our reads come from a control dataset with no splicing.

There should be three more files in the `data` directory now: `30xpolyA.bam`, `30xpolyA.sorted.bam`, and `30xpolyA.sorted.bam.bai`.

### 5.5 Segmentation and tail length estimation with nanopolish polya

Finally, we can run the polyadenylation estimator:

```
nanopolish polya --threads=8 --reads=data/30xpolyA.fastq --bam=data/30xpolyA.sorted.bam --genome=data/enolase_reference.fas > polya_results.tsv
```

Set the `--threads` flag to the number of parallel threads you want to use. Generally speaking, a larger number of threads tends to lower the compute time, but there are diminishing returns to a higher value and performance can actually decrease if your CPU is incapable of supporting your desired number of parallel threads. The best number of threads to use is highly dependent upon your hardware.

### 5.6 Interpreting the output TSV file

We’ll end this quickstart with a look at the output of the polya program. Let’s look at the top five lines of the `polya_results.tsv` file we’ve just generated:

```
readname  contig  position leader_start adapter_start polya_start transcript_start read_rate polya_length qc_tag
```

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d6f42b79-90c6-4edd-8c8f-8a7ce0ac6ecb</td>
<td>YHR174W</td>
<td>0</td>
<td>54.0</td>
<td>3446.0</td>
<td>7216.0</td>
<td>38.22</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>e02d9858-0c04-4d86-8dba-18a47d9ac005</td>
<td>YHR174W</td>
<td>0</td>
<td>221.0</td>
<td>1812.0</td>
<td>11046.0</td>
<td>27.48</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>b588dee2-2c5b-410c-81a6-8576e23390ed</td>
<td>YHR174W</td>
<td>0</td>
<td>22.0</td>
<td>8338.0</td>
<td>14294.0</td>
<td>32.43</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>af9dfee2-1711-4083-b109-487b99895e0a</td>
<td>YHR174W</td>
<td>0</td>
<td>889.0</td>
<td>3679.0</td>
<td>14924.0</td>
<td>32.43</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>93f98a86-3b18-48cf-8c4d-15cf27791e2</td>
<td>YHR174W</td>
<td>0</td>
<td>144.0</td>
<td>1464.0</td>
<td>6140.0</td>
<td>39.65</td>
<td>PASS</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
Each row corresponds to the output for a given read. The columns have the following interpretation: *readname* refers to the unique ID associated to this particular read. This string is also used to look up the corresponding fast5 file, e.g. by looking through the *readdb* file generated by *nanopolish index*.

- **contig** refers to the reference sequence that this read aligns to, and is taken from the BAM file.
- **position** is the 5’ starting position of the alignment to the reference sequence, and also comes from the BAM file.
- **leader_start**, **adapter_start**, **polya_start**, and **transcript_start** are the indices of the signal segmentation generated by the underlying model within *nanopolish*. Briefly, there are four biologically-meaningful regions of the raw sequence of electrical current readings within each fast5 file; these four entries denote the starting index of each of these consecutive regions. The indices start from 0 and are oriented in the 3’-to-5’ direction (due to the inherent orientation of the native RNA nanopore sequencing protocol). A full exposition of this segmentation algorithm is available in the ‘supplementary note’ to our associated publication.
- **read_rate** is the estimated translocation rate (in units of nucleotides/second) of the read through the pore. The translocation rate is non-uniform during the sequencing process of even a single molecule, so this is ultimately a summary statistic of the dynamic, time-varying rate.
- **polya_length** is the estimated polyadenylated tail length, in number of nucleotides. That this value is a float rather than an integer reflects the fact that our estimated tail length is the output of an estimator based on the translocation rate.
- **qc_tag** is an additional flag used to indicate the validity of the estimate. Generally speaking, you should only use rows of the output file with this value set to *PASS*; all other rows with (e.g.) the *qc_tag* set to *SUFFCLIP*, *ADAPTER*, etc. display signs of irregularity that indicate that we believe the estimate to be unreliable. You can easily filter away all rows with the tag set to anything other than *PASS* using a *grep*:

```
grep 'PASS' polya_results.tsv > polya_results.pass_only.tsv
```
CHAPTER 6

Helping us debug nanopolish

6.1 Overview

Running into errors with nanopolish? To help us debug, we need to be able to reproduce the errors. We can do this by packaging a subset of the files that were used by a nanopolish. We have provided scripts/extract_readsAligned_to_region.py and this tutorial to help you do exactly this.

Briefly, this script will:

• extract reads that align to a given region in the draft genome assembly
• rewrite a new BAM, BAI, FASTA file with these reads
• extract the FAST5 files associated with these reads
• save all these files into a tar.gz file

6.2 Workflow

1. Narrow down a problematic region by running nanopolish variants --consensus [...] and changing the -w parameter.
2. Run the scripts/extract_reads_aligned_to_region.py.
3. Send the resulting tar.gz file to us by hosting either a dropbox or google drive.

6.3 Tutorial - using extraction helper script to create example datasets

We extracted a subset of reads for a 2kb region to create the example dataset for the eventalign and consensus tutorial using scripts/extract_reads_aligned_to_region.py. Here is how:
Generated basecalled --reads file:

1. Basecalled reads with albacore:

```bash
read_fast5_basecaller.py -c r94_450bps_linear.cfg -t 8 -i /path/to/raw/fast5/files -s /path/to/albacore-2.0.1/output/directory -o fastq
```

2. Merged the different albacore fastq outputs:

```bash
cat /path/to/albacore-2.0.1/output/directory/workspace/pass/*.fastq > albacore-2.0.1-merged.fastq
```

3. Converted the merged fastq to fasta format:

```bash
paste - - - - < albacore-2.0.1-merged.fastq | cut -f 1,2 | sed 's/^@/>/' | tr "\t" "\n" > reads.fasta
```

Generated --bam file with the draft genome assembly (--g):

1. Ran canu to create draft genome assembly:

```bash
canu \
    -p ecoli -d outdir genomeSize=4.6m \
    -nanopore-raw reads.fasta \
```

2. Index draft assembly:

```bash
bwa index ecoli.contigs.fasta
samtools faidx ecoli.contigs.fasta
```

3. Aligned reads to draft genome assembly with bwa (v0.7.12):

```bash
bwa mem -x ont2d ecoli.contigs.fasta reads.fasta | samtools sort -o reads.sorted.bam | samtools index reads.sorted.bam
```

Selected a --window:

1. Identified the first contig name and chose a random start position:

```bash
head -3 ecoli.contigs.fasta
```

Output:
As we wanted a 2kb region, we selected a random start position (200000) so our end position was 202000. Therefore our --window was “tig00000001:200000-202000”.

Using the files we created, we ran scripts/extract_reads_aligned_to_region.py, please see usage example below.

**Note:** Make sure nanopolish still reproduces the same error on this subset before sending it to us.

### 6.4 Usage example

```
python3 extract_reads_aligned_to_region.py \
--reads reads.fasta \
--genome ecoli.contigs.fasta \
--bam reads.sorted.bam \
--window "tig00000001:200000-202000" \
--output_prefix ecoli_2kb_region --verbose
```

<table>
<thead>
<tr>
<th>Argument name(s)</th>
<th>Req.</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b, --bam</td>
<td>Y</td>
<td>NA</td>
<td>Sorted bam file created by aligning reads to the draft genome.</td>
</tr>
<tr>
<td>-g, --genome</td>
<td>Y</td>
<td>NA</td>
<td>Draft genome assembly</td>
</tr>
<tr>
<td>-r, --reads</td>
<td>Y</td>
<td>NA</td>
<td>Fasta, fastq, fasta.gz, or fastq.gz file containing base-called reads.</td>
</tr>
<tr>
<td>-w, --window</td>
<td>Y</td>
<td>NA</td>
<td>Draft genome assembly coordinate string ex. “contig:start-end”. It is essential that you wrap the coordinates in quotation marks (“”).</td>
</tr>
<tr>
<td>-o, --output_prefix</td>
<td>N</td>
<td>reads_subset</td>
<td>Prefix of output tar.gz and log file.</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>N</td>
<td>False</td>
<td>Use for verbose output with info on progress.</td>
</tr>
</tbody>
</table>

### 6.5 Script overview

1. Parse input files
2. Assumes readdb file name from input reads file
3. Validates input

   - checks that input bam, readdb, fasta/q, draft genome assembly, draft genome assembly index file exist, are not empty, and are readable

---

6.4. Usage example 23
4. With user input draft genome assembly coordinates, extracts all reads that aligned within these coordinates stores the read_ids.
   - uses pysam.AlignmentFile
   - uses samfile.fetch(region=draft_ga_coords) to get all reads aligned to region
   - if reads map to multiple sections within draft ga it is not added again

5. Parses through the input readdb file to find the FAST5 files associated with each region that aligned to region
   - stores in dictionary region_fast5_files; key = read_id, value = path/to/fast5/file
   - path to fast5 file is currently dependent on the user’s directory structure

6. Make a BAM and BAI file for this specific region
   - creates a new BAM file called region.bam
   - with pysam.view we rewrite the new bam with reads that aligned to the region…
   - with pysam.index we create a new BAI file

7. Now to make a new FASTA file with this subset of reads
   - the new fasta file is called region.fasta
   - this first checks what type of sequences file is given { fasta, fastq, fasta.gz, fastq.gz }
   - then handles based on type of seq file using SeqIO.parse
   - then writes to a new fasta file

8. Let’s get to tarring
   - creates a tar.gz file with the output prefix
   - saves the fast5 files in directory output_prefix/fast5_files/

9. Adds the new fasta, new bam, and new bai file with the subset of reads
10. Adds the draft genome assembl and associated fai index file
11. Performs a check
    - the number of reads in the new BAM file, new FASTA file, and the number of files in the fast5_files directory should be equal
12. Outputs a tar.gz and log file. FIN!
CHAPTER 7

Manual

Modules available:

- `nanopolish extract`: extract reads in FASTA or FASTQ format from a directory of FAST5 files
- `nanopolish call-methylation`: predict genomic bases that may be methylated
- `nanopolish variants`: detect SNPs and indels with respect to a reference genome
- `nanopolish variants --consensus`: calculate an improved consensus sequence for a draft genome assembly
- `nanopolish eventalign`: align signal-level events to k-mers of a reference genome
- `nanopolish phase-reads`: Phase reads using heterozygous SNVs with respect to a reference genome
- `nanopolish polya`: Estimate polyadenylated tail lengths on native RNA reads

7.1 extract

7.1.1 Overview

This module is used to extract reads in FASTA or FASTQ format from a directory of FAST5 files.

7.1.2 Input

- path to a directory of FAST5 files modified to contain basecall information

7.1.3 Output

- sequences of reads in FASTA or FASTQ format
### 7.1.4 Usage example

```
nanopolish extract [OPTIONS] <fast5|dir>
```

<table>
<thead>
<tr>
<th>Argument name(s)</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;fast5</td>
<td>dir&gt;</td>
<td>Y</td>
<td>NA</td>
</tr>
<tr>
<td>-r, --recurse</td>
<td>N</td>
<td>NA</td>
<td>Recurse into subdirectories</td>
</tr>
<tr>
<td>-g, --fastq</td>
<td>N</td>
<td>fasta format</td>
<td>Use when you want to extract to FASTQ format</td>
</tr>
<tr>
<td>-t, --type=TYPE</td>
<td>N</td>
<td>2d-or-template</td>
<td>The type of read either: {template, complement, 2d, 2d-or-template, any}</td>
</tr>
<tr>
<td>-b, --basecaller=NAME[:VERSION]</td>
<td>N</td>
<td>NA</td>
<td>consider only data produced by basecaller NAME, optionally with given exact VERSION</td>
</tr>
<tr>
<td>-o, --output=FILE</td>
<td>N</td>
<td>stdout</td>
<td>Write output to FILE</td>
</tr>
</tbody>
</table>

### 7.2 index

#### 7.2.1 Overview

Build an index mapping from basecalled reads to the signals measured by the sequencer

#### 7.2.2 Input

- path to directory of raw nanopore sequencing data in FAST5 format
- basecalled reads

#### 7.2.3 Output

- gzipped FASTA file of basecalled reads (.index)
- index files (.fai, .gzi, .readdb)

#### 7.2.4 Readdb file format

Readdb file is a tab-separated file that contains two columns. One column represents read ids and the other column represents the corresponding path to FAST5 file:

```
read_id_1  /path/to/fast5/containing/reads_id_1/signals
read_id_2  /path/to/fast5/containing/read_id_2/signals
```
7.2.5 Usage example

```
nanopolish index [OPTIONS] -d nanopore_raw_file_directory reads.fastq
```

<table>
<thead>
<tr>
<th>Argument name(s)</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d, --directory</td>
<td>Y</td>
<td>NA</td>
<td>FAST5 or path to directory of FAST5 files containing ONT sequencing raw signal information.</td>
</tr>
<tr>
<td>-f, --fast5-fofn</td>
<td>N</td>
<td>NA</td>
<td>file containing the paths to each fast5 for the run</td>
</tr>
</tbody>
</table>

7.3 call-methylation

7.3.1 Overview

Classify nucleotides as methylated or not.

7.3.2 Input

- Basecalled ONT reads in FASTA format

7.3.3 Output

- tab-separated file containing per-read log-likelihood ratios

7.3.4 Usage example

```
nanopolish call-methylation [OPTIONS] <fast5|dir>
```

<table>
<thead>
<tr>
<th>Argument name(s)</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r, --reads=FILE</td>
<td>Y</td>
<td>NA</td>
<td>the ONT reads are in fasta FILE</td>
</tr>
<tr>
<td>-b, --bam=FILE</td>
<td>Y</td>
<td>NA</td>
<td>the reads aligned to the genome assembly are in bam FILE</td>
</tr>
<tr>
<td>-g, --genome=FILE</td>
<td>Y</td>
<td>NA</td>
<td>the genome we are computing a consensus for is in FILE</td>
</tr>
<tr>
<td>-t, --threads=NUM</td>
<td>N</td>
<td>1</td>
<td>use NUM threads</td>
</tr>
<tr>
<td>--progress</td>
<td>N</td>
<td>NA</td>
<td>print out a progress message</td>
</tr>
</tbody>
</table>
7.4 variants

7.4.1 Overview

This module is used to call single nucleotide polymorphisms (SNPs) using a signal-level HMM.

7.4.2 Input

- basecalled reads
- alignment info
- genome assembly

7.4.3 Output

- VCF file

7.4.4 Usage example

```
nanopolish variants [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome.fa
```
### Argument name(s) | Required | Default value | Description
---|---|---|---
--snps | N | NA | use flag to only call SNPs
--consensus=FILE | N | NA | run in consensus calling mode and write polished sequence to FILE
--fix-homopolymers | N | NA | use flag to run the experimental homopolymer caller
--faster | N | NA | minimize compute time while slightly reducing consensus accuracy
--window=STR | N | NA | find variants in window STR (format: <chromosome_name>:<start>-<end>)
--reads=FILE | Y | NA | the ONT reads are in fasta FILE
--bam=FILE | Y | NA | the reads aligned to the reference genome are in bam FILE
--event-bam=FILE | Y | NA | the events aligned to the reference genome are in bam FILE
--genome=FILE | Y | NA | the reference genome is in FILE
--outfile=FILE | N | stdout | write result to FILE
--threads=NUM | N | 1 | use NUM threads
--min-candidate-frequency=F | N | 0.2 | extract candidate variants from the aligned reads when the variant frequency is at least F
--min-candidate-depth=D | N | 20 | extract candidate variants from the aligned reads when the depth is at least D
--max-haplotypes=N | N | 1000 | consider at most N haplotypes combinations
--max-rounds=N | N | 50 | perform N rounds of consensus sequence improvement
--candidates=VCF | N | NA | read variants candidates from VCF, rather than discovering them from aligned reads
--alternative-basecalls-bam=FILE | N | NA | if an alternative basecaller was used that does not output event annotations then use basecalled sequences from FILE. The signal-level events will still be taken from the -b bam
--calculate-all-support | N | NA | when making a call, also calculate the support of the 3 other possible bases
--models-fofn=FILE | N | NA | read alternatives k-mer models from FILE

### 7.5 event align

#### 7.5.1 Overview
Align nanopore events to reference k-mers

#### 7.5.2 Input
- basecalled reads
- alignment information
• assembled genome

## 7.5.3 Usage example

```bash
nanopolish eventalign [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome. →fa
```

<table>
<thead>
<tr>
<th>Argument name(s)</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--sam</td>
<td>N</td>
<td>NA</td>
<td>use to write output in SAM format</td>
</tr>
<tr>
<td>-w, --window=STR</td>
<td>N</td>
<td>NA</td>
<td>Compute the consensus for window STR (format: ctg:start_id-end_id)</td>
</tr>
<tr>
<td>-r, --reads=FILE</td>
<td>Y</td>
<td>NA</td>
<td>the ONT reads are in fasta FILE</td>
</tr>
<tr>
<td>-b, --bam=FILE</td>
<td>Y</td>
<td>NA</td>
<td>the reads aligned to the genome assembly are in bam FILE</td>
</tr>
<tr>
<td>-g, --genome=FILE</td>
<td>Y</td>
<td>NA</td>
<td>the genome we are computing a consensus for is in FILE</td>
</tr>
<tr>
<td>-t, --threads=NUM</td>
<td>N</td>
<td>1</td>
<td>use NUM threads</td>
</tr>
<tr>
<td>--scale-events</td>
<td>N</td>
<td>NA</td>
<td>scale events to the model, rather than vice-versa</td>
</tr>
<tr>
<td>--progress</td>
<td>N</td>
<td>NA</td>
<td>print events to the model, rather than vice-versa</td>
</tr>
<tr>
<td>-n, --print-read-names</td>
<td>N</td>
<td>NA</td>
<td>print read names instead of indexes</td>
</tr>
<tr>
<td>--summary=FILE</td>
<td>N</td>
<td>NA</td>
<td>summarize the alignment of each read/strand in FILE</td>
</tr>
<tr>
<td>--samples</td>
<td>N</td>
<td>NA</td>
<td>write the raw samples for the event to the tsv output</td>
</tr>
<tr>
<td>--models-fofn=FILE</td>
<td>N</td>
<td>NA</td>
<td>read alternative k-mer models from FILE</td>
</tr>
</tbody>
</table>

## 7.6 phase-reads - (experimental)

### 7.6.1 Overview

Phase reads using heterozygous SNVs with respect to a reference genome

### 7.6.2 Input

- basecalled reads
- alignment information
- assembled genome
- variants (from nanopolish variants or from other sources eg. Illumina VCF)

### 7.6.3 Usage example

```bash
nanopolish phase-reads [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome. →fa variants.vcf
```
7.7 polya

7.7.1 Overview

Estimate the number of nucleotides in the poly(A) tails of native RNA reads.

7.7.2 Input

- basecalled reads
- alignment information
- reference transcripts

7.7.3 Usage example

```
nanopolish polya [OPTIONS] --reads=reads.fa --bam=alignments.bam --genome=ref.fa
```


CHAPTER 9

Credits and Thanks

The fast table-driven logsum implementation was provided by Sean Eddy as public domain code. This code was originally part of hmmer3. Nanopolish also includes code from Oxford Nanopore’s scrappie basecaller. This code is licensed under the MPL.