NYGC high coverage sequence dataset

The high coverage dataset used in the present study consists of 3202 samples sequenced to 30X coverage by NYGC. The dataset includes 2504 samples of the original 1000 Genomes Project's phase three panel (available in ENA as study ERP114329, https://www.ebi.ac.uk/ena/browser/view/PRJEB31736) plus 698 further samples related to the main phase 3 panel (available in ENA under study ERP120144, https://www.ebi.ac.uk/ena/browser/view/PRJEB36890).

Alignment generation pipeline used by NGYC

The high coverage dataset sequences were produced using PCR-free sequencing libraries sequenced Illumina NovaSeq 6000 platform with 150bp paired-end reads. Reads were aligned to hs38DH human reference genome (GRCh38 assembly with additional decoy sequences and HLA genes) available at ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/GRCh38_reference_genome/GRCh38_full_analysis_set_plus_decoy_hla.fa. The initial alignments at lane level were generated using BWA-MEM algorithm (Li, 2013) (https://github.com/lh3/bwa, v0.7.15), with Piccard tools (https://broadinstitute.github.io/picard, v2.4.1) then used to fix mate-pair information, merge lane-level BAM files into a single sample-level file, mark duplicate reads and coordinate sort the sample-level BAM file. Base quality scores were recalibrated using known SNPs with GATK (https://github.com/broadinstitute/gatk/, v3.5), and the final alignment files in CRAM format with their associated CRAI index files generated using Samtools (https://www.htslib.org/, v1.3.1) and.
These final alignment files, available on the 1000genomes.ebi.ac.uk ftp server at https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/ are the source of the high coverage mitochondrial sequence data we use in the present study.

For the full details of the NYGC high coverage dataset sequence alignment and processing pipeline see https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/20190405_NYGC_b38_pipeline_description.pdf

Sequence alignment index files

High coverage sequences of the 2504 samples of the 1000 Genomes Project's phase 3 panel:
https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/1000G_2504_high_coverage.sequence.index
High coverage sequences of the additional 698 related samples:
https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/1000G_698_related_high_coverage.sequence.index

High coverage MT sequence data retrieval and initial processing: Samtools

The high coverage mitochondrial sequence dataset used in the present study was obtained by extracting the reads mapped to the MT genome from the 3202 high coverage whole genome alignment CRAM files made available by NYGC. We used SAMtools v1.7 to extract the mitochondrial reads and generate pileup format files for each sample from the subset of MT reads passing our chosen minimum base call and mapping quality criteria.

Pipeline

STEP 1: Retrieve reads mapped to MT genome and save locally in BAM format

```
samtools view -b -X path/to/remote/CRAM path/to/remote/CRAI chrM:1-16569 -o output.bam
```

Inputs

`path/to/remote/CRAM`	Full alignment in CRAM format
`path/to/remote/CRAI`	The corresponding alignment index file in CRAI format

Outputs

`output.bam`	Extracted mitochondrial alignment in BAM format

Options

`chrM:1-16569`	Set region of interest to full MT genome
`-b`	Specify output in BAM format
`-X`	Specify alignment index file location

STEP 2: Index the created BAM file

```
samtools index -b input.bam
```

Inputs

`input.bam`	Mitochondrial read alignment in BAM format

Outputs

`input.bam.bai`	Index file saved in the same directory as the input BAM file

STEP 3: Generate pileup format TXT file from reads passing specified quality control criteria

```
samtools mpileup -f rCRS.fa -q25 -Q30 -d0 --incl-flags 99,147,83,163 -a input.bam -o output.txt;
```

Inputs

`rCRS.fa`	rCRS reference sequence in FASTA format
`input.bam`	Mitochondrial read alignment in BAM format

Outputs

`output.txt`	Text file in pileup format

Settings

`-Q30`	Set minimum base quality (BAQ): 30
`-q25`	Set minimum read mapping quality (MAPQ): 25
`-d0`	Remove limit on maximum number reads to include
`--incl-flags 99,147,83,163`	Only include properly mapped read pairs where both reads are mapped in correct orientation with correct insert size
`-a`	Output all positions, including those with zero depth

CONSENSUS CALLING: Generated mitochondrial sequence alignment BAM files were also used to call sample consensus sequences. We used Samtools version 1.15, with simple consensus calling mode and applying the same SAM flag and read mapping quality score thresholds as used for pileup generation. In order to maintain the same consensus sequence length as rCRS any deletions were included while insertions were excluded.

STEP 1: Call consensus sequence for each sample

```
samtools consensus -m simple -aq -r chrM:1-16569 --show-ins no --show-del yes
--rf 99,147,83,163 --min-MQ 25 -f fasta input.bam -o cons_seq.fa
```

Inputs

`input.bam`	Sample sequence alignment in BAM format

Outputs

cons_seq.fa	Sample consensus sequence in FASTA format

Options

`-m simple`	Use simple consensus calling mode (frequency counting algorithm)
`-r chrM:1-16569`	Include all MT genome positions
`-a`	Output all bases in region
`-q`	Use base quality scores
`--min-MQ 25`	Only use reads with mapping quality 25 or higher
`--incl-flags 99,147,83,163`	Only include properly mapped read pairs where both reads are mapped in correct orientation with correct insert size
`--show-del yes`	Include deletions as `'*'`
`--show-ins no`	Exclude insertions
`-f fasta`	Specify output in FASTA format

High coverage MT sequence pileup parsing: Python

Pileup format

Pileup is a tab delimited text format where each line represents one sequence position and consists of 5 columns: (1) sequence identifier (str), (2) position number (int), (3) reference base (char), (4) number of reads covering the position (int), (5) bases called at this position (str) and, optionally, (6) base quality scores at this position (str).

The encoding used in the base call string:

Forward direction	Reverse direction	Interpretation
.	,	Reference match
ACTGN	actgn	Reference mismatch
^		Start of a read marker
Any single character following '^'		Read mapping quality as ASCII value -33
$		End of a read marker
*		Placeholder for 2nd deleted base onwards in a multi-bp deletion
+[length][sequence_inserted]		Insertion after this base
-[length][sequence_deleted]		Deletion after this base

STEP 1: Parse each sample pileup file to obtain a tally of all possible base calls found at any given MT genome position in that sample.

Script: `Pileup_parsing.py`

Inputs

`in_file.txt`	Sample-level sequence alignment file in pileup format
`ref_file.txt`	Text file with MT reference sequence (rCRS)

Outputs

`out_file.txt`	Tab-delimited output .txt file with position data and parsed read counts organised in 21 columns (headers: "Chromosome", "Position", "Reference", "Reads", "Total_As", "Total_Cs", "Total_Ts", "Total_Gs", "Total_Ns", "For_As", "Rev_As", "For_Cs", "Rev_Cs", "For_Ts", "Rev_Ts", "For_Gs", "Rev_Gs", "For_Ns", "Rev_Ns", "Insertions", "Deletions"), with each line corresponding to one mtDNA position

Algorithm structure

- Load rCRS reference sequence from `ref_file.txt` as a string
- Create empty output file `out_file.txt` and write the first line of column headers
- Read the input pileup file `in_file.txt` line by line. For each line:
  - Split the line into the Chromosome ID, Position, Reference base, Read count and a Pileup string
  - Parse the pileup string one character at a time as follows
    - Forward reference match (`"."`) – Increment the relevant forward base counter
    - Forward reference mismatch (`'A'/'C'/'G'/'T'/'N'`)- Increment the corresponding forward counter
    - Reverse reference match (`","`) – Increment the relevant reverse base counter
    - Reverse reference mismatch (`'a'/'c'/'g'/'t'/'n'`) - Increment the corresponding reverse counter
    - Start of an indel (`"+"/"-"`) – increment the insertion (`"+"`) or deletion (`"-"`) counter and move pointer forward to skip the indel sequence
    - Base in a multi-bp deletion (`'*'`) – increment the counter for deletions
    - Start of read marker (`"$"`) – move pointer to skip the character
    - End of read marker (`"^"`) – move pointer to skip this and the following quality score character
  - Assemble tab delimited output string containing position number, reference base, total read count and the parsed counts of different base calls and indels and write it as a new line into the `out_file.txt`
- If the pileup file is missing data for any positions, the reference base is read from rCRS and a line with 0s for all counters written to the output file for that position

STEP 2: Aggregate base count data from individual samples into a population level data files such that each output data file contains an array of base call counts with one row per sample and one column per mtDNA position.

Script: `Merge_counts.py`

Inputs

`[SampleID]_basecounts.txt`	Multiple tab-delimited text files generated by `Pileup_parsing.py`, each listing founts of different types of base calls recorded at each mtDNA position in a specific sample

Outputs: 19 tab delimited `.txt` files

`Sample_IDs.txt`	List of the processed sample IDs

`For_As.txt`
`For_Cs.txt`
`For_Gs.txt`
`For_Ts.txt`
`For_Ns.txt`
	Counts of A/C,G,T or N calls in forward sequencing at each mtDNA position (columns) in each sample (rows)

`Rev_As.txt`
`Rev_Cs.txt`
`Rev_Gs.txt`
`Rev_Ts.txt`
`Rev_Ns.txt`
	Counts of A/C,G,T or N calls in reverse sequencing direction at each mtDNA position (columns) in each sample (rows)

`All_As.txt`
`All_Cs.txt`
`All_Gs.txt`
`All_Ts.txt`
`All_Ns.txt`
	Total counts of A/C,G,T or N calls across both sequencing directions at each mtDNA position (columns) in each sample (rows)

`Insertions.txt` `Deletions.txt`	Number of reads recording insertions or deletions at each mtDNA position (columns) in each sample (rows)
`Reads.txt`	Total analysed reads at each position (columns) in each sample (rows)

Algorithm structure

> Create output directory
> Create 19 empty output files for recording sample IDs (`Sample_IDs.txt`), total read counts (`Reads.txt`), indel counts (`Insertions.txt`, `Deletions.txt`), and different base call counts in forward direction (`For_As.txt`, `For_Cs.txt`, `For_Gs.txt`, `For_Ts.txt`, `For_Ns.txt`), reverse direction (`Rev_As.txt`, `Rev_Cs.txt`, `Rev_Gs.txt`, `Rev_Ts.txt`, `Rev_Ns.txt`) and across both directions (`All_As.txt`, `All_Cs.txt`, `All_Gs.txt`, `All_Ts.txt`, `All_Ns.txt`), and write header line to each output file file.
> Access the input directory and loop through sample-level base count data files, processing each one at a time:
>> Extract sample ID from file name
>> Read sample-level base count data file columns into separate arrays
>> Insert sample ID at the start of each column-derived base count array and write the array as tab-delimited string into a new line of the appropriate output file
>> Append the column-derived data arrays as a tab delimited strings to the corresponding output files as new lines.
>> Append sample ID to `Sample_IDs.txt` file

High coverage MT sequence allele frequency calculation: MATLAB

Step 1: **Import read count data**

Load read count matrices from .txt files. Transpose all variables such that positions = rows and samples = columns. Save all produced variables into .mat files.

Code: SCRIPT_1_Basecount_import.m

Variables generated in step 1			**Dimensions & data type**		
SampleIDs			1 x 3202	string	7 letter G1K project sample IDs
Individuals			1 x 3202	double	Numeric IDs (original processing order)
Positions			16569 x 1	double	mtDNA position numbers
Reads_TOTAL			16569 x 3202	double	Total reads at each position in each sample
Basecalls	Forward	A	16569 x 3202	double	Base counts at each position in each sample. Rows – Positions Columns – Samples
		C	16569 x 3202	double	
		G	16569 x 3202	double	
		T	16569 x 3202	double	
		N	16569 x 3202	double	
	Reverse	A	16569 x 3202	double	
		C	16569 x 3202	double	
		G	16569 x 3202	double	
		T	16569 x 3202	double	
		N	16569 x 3202	double	
	All	A	16569 x 3202	double	
		C	16569 x 3202	double	
		G	16569 x 3202	double	
		T	16569 x 3202	double	
		N	16569 x 3202	double	
Indels	Insertions		16569 x 3202	double	Indel counts at each position in each sample
	Deletions		16569 x 3202	double	

Basecall and Indel variables grouped into higher level data structures

Step 2: **Import MT reference and sample consensus sequences**

Import from FASTA file rCRS reference sequence and 3202 sample consensus sequences. Apply following modifications to consensus sequences:
(1) replace all deletions marked as '*' with '−';
(2) set position 3107 as '−';
(3) change all letters to be upper case;
(4) set any non- A/C/G/T/N/− characters as 'N'.

Code: SCRIPT_2_consensus_seq_import.m

Variables generated in step 2	**Dimensions & data type**		
ConsensusSeqs	16569 x 3202	char	Sample consensus sequences
rCRS	16569 x 1	char	rCRS sequence

Step 3: Calculate total and consensus base frequencies

Excluding indels, calculate total read counts and base frequencies. Identify excess and consensus read fractions, calculate consensus and excess read counts and consensus base frequencies.

Code: SCRIPT_3_Base_frequency_calculation.m

Variables generated in step 3			**Dimensions & data type**	
Reads	Forward		*3 variables:* 16569 x 3202 double	Sum of all A, C, G, T base calls
	Reverse			
	All			
Frequencies	Forward	A,C,G,T	*3 x 4 variables:* 16569 x 3202 double	Proportion of each base type among the sum base calls
	Reverse	A,C,G,T		
	All	A,C,G,T		
Consensus_basecalls	Forward	A,C,G,T	*3 x 4 variables:* 16569 x 3202 double	Calls of each base type with proportional support in the opposite direction
	Reverse	A,C,G,T		
	All	A,C,G,T		
Excess_basecalls	Forward	A,C,G,T	*3 x 4 variables:* 16569 x 3202 double	Excess calls of each base type without matching support in the opposite direction
	Reverse	A,C,G,T		
	All	A,C,G,T		
Consensus_reads	Forward		*3 variables:* 16569 x 3202 double	Sum of A, C, G, T consensus base calls
	Reverse			
	All			
Excess_reads	Forward		*3 variables:* 16569 x 3202 double	Sum of A, C, G, T excess base calls
	Reverse			
	All			
Consensus_frequencies		A	*4 variables:* 16569 x 3202 double	Proportion of each base type among sum consensus base calls. Consensus frequencies are the same between directions.
		C		
		G		
		T		

STANDARD METHOD: Step 4: Sort alleles by total frequencies

Alleles are sorted by total base call count in decreasing frequency order as Major allele, Minor allele 1, Minor allele 2 and Minor allele 3.

Code: SCRIPT_4_Allele_sorting_by_raw_frequency.m

Variables generated in step 4			**Dimensions & data type**	
Allele_count			16569 x 3202 double	Number of alleles found at each site
Allele_order	A		4 variables 16569 x 3202 double	Index where in the allele order by total read count each base is found: 1 = major allele, 2 = minor allele 1, 3 = minor allele 2, 4 = minor allele 3
	C			
	G			
	T			
Allele_bases	Major		4 variables 16569 x 3202 char	Matrices listing bases of each of the sorted alleles
	Minor_1			
	Minor_2			
	Minor_3			
Allele_reads	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Base calls supporting each of the sorted alleles. Generated from "Reads" re-organised by "Allele_order"
	Minor_1	For, Rev, All		
	Minor_2	For, Rev, All		
	Minor_3	For, Rev, All		
Allele_ frequencies	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Frequencies of each of the sorted alleles. Produced by re-organising "Frequencies" in "Allele_order"
	Minor_1	For, Rev, All		
	Minor_2	For, Rev, All		
	Minor_3	For, Rev, All		

STANDARD METHOD: **Step 5: Heteroplasmy identification and filtering based on allele total frequencies**

7 different filtration stringency steps based on allele total read counts and raw frequencies are applied. Filtered datasets of alleles passing each step are saved.

Code: SCRIPT_5_ Heteroplasmy_filtering_STANDARD_METHOD.m

Variables generated in step 5				**Dimensions & data type**	
Heteroplasmy_filters				7 x 5 table	Table listing settings for each filter
Variable set for each filter	Allele_masks	Major		4 variables 16569 x 3202 logical	Masks for allele filter status: TRUE = allele passes filter, FALSE = allele fails filter
		Minor_1			
		Minor_2			
		Minor_3			
	Allele_count			16569 x 3202 double	Alleles found passing filter at each site
	Allele_bases	Major		4 variables 16569 x 3202 char	Base types of major and minor alleles that passed the relevant filter
		Minor_1			
		Minor_2			
		Minor_3			
	Allele_reads	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Base calls supporting each allele that passed the relevant filter
		Minor_1	For, Rev, All		
		Minor_2	For, Rev, All		
		Minor_3	For, Rev, All		
	Allele_ _frequencies	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Frequencies of each allele that passed the relevant filter
		Minor_1	For, Rev, All		
		Minor_2	For, Rev, All		
		Minor_3	For, Rev, All		
	Heteroplasmic_sites			16569 x 3202 logical	TRUE = 2 or more alleles pass filter
	Homoplasmic_sites			16569 x 3202 logical	TRUE = exactly 1 allele passes filter

CONSENSUS METHOD: Step 6: Sort alleles by consensus frequencies

Alleles are sorted by consensus read count in decreasing frequency order as Major and Minor 1, 2 and 3.

Code: SCRIPT_6_Allele_sorting_by_consensus_frequency.m

Variables generated in step 5			**Dimensions & data type**	
Allele_count			16569 x 3202 double	Number of alleles found at each site
Allele_order	A		4 variables 16569 x 3202 double	Index where in the allele order each base is by consensus read count: 1 = major allele, 2 = minor allele 1, 3 = minor allele 2, 4 = minor allele 3
	C			
	G			
	T			
Allele_bases	Major		4 variables 16569 x 3202 char	Matrices listing base type of each of the sorted alleles
	Minor_1			
	Minor_2			
	Minor_3			
Allele_consensus_ _reads	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Consensus base calls supporting each allele. Produced by re-organising "Consensus_reads" in "Allele_order"
	Minor_1	For, Rev, All		
	Minor_2	For, Rev, All		
	Minor_3	For, Rev, All		
Allele_excess_reads	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Excess base calls supporting each allele. Produced by re-organising "Excess_reads" in "Allele_order"
	Minor_1	For, Rev, All		
	Minor_2	For, Rev, All		
	Minor_3	For, Rev, All		
Allele_consensu_ _frequencies	Major		4 variables: 16569 x 3202 double	Consensus frequencies of the sorted alleles. Produced by re-organising "Consensus_frequencies" in "Allele_order"
	Minor_1			
	Minor_2			
	Minor_3			

CONSENSUS METHOD: Step 7: Heteroplasmy identification and filtering based on allele consensus frequencies

7 different filtration stringency steps based on allele consensus read counts and consensus frequencies are applied. Filtered datasets of alleles passing each step are saved in separate sets.

Code: SCRIPT_7_Heteroplasmy_filtering_CONSENSUS_METHOD.m

Variables generated in step 7				**Dimensions & data type**	
Heteroplasmy_filters				7 x 5 table	Table listing settings for each filter
Variable set for each filter	Allele_masks	Major		4 variables 16569 x 3202 logical	Masks for allele filter status: TRUE = allele passes filter, FALSE = allele fails filter
		Minor_1			
		Minor_2			
		Minor_3			
	Allele_count			16569 x 3202 double	Alleles found passing filter at each site
	Allele_bases	Major		4 variables 16569 x 3202 char	Base types of major and minor alleles that passed the relevant filter
		Minor_1			
		Minor_2			
		Minor_3			
	Allele_ _consensus_ _reads	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Consensus base calls supporting each filter-passing allele
		Minor_1	For, Rev, All		
		Minor_2	For, Rev, All		
		Minor_3	For, Rev, All		
	Allele_ _excess_ _reads	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Excess base calls supporting each filter-passing allele
		Minor_1	For, Rev, All		
		Minor_2	For, Rev, All		
		Minor_3	For, Rev, All		
	Allele_ _consensus_ _frequencies	Major	For, Rev, All	4 x 3 variables: 16569 x 3202 double	Consensus frequencies of each allele that passed the relevant filter
		Minor_1	For, Rev, All		
		Minor_2	For, Rev, All		
		Minor_3	For, Rev, All		
	Heteroplasmic_sites			16569 x 3202 logical	TRUE = 2 or more alleles pass filter
	Homoplasmic_sites			16569 x 3202 logical	TRUE = exactly 1 allele passes filter

Allele filtration settings used in STEPS 5 and 7

Filter ID	Reads supporting allele		Allele frequency	
	Overall	Per direction	Overall	Per direction
Filter 1	1	0	0	0
Filter 2	2	1	0	0
Filter 3	10	5	0	0
Filter 4	10	5	0.1%	0
Filter 5	10	5	0.1%	0.1%
Filter 6	10	5	1%	0
Filter 7	10	5	1%	1%
Standard method: All base calls and allele raw frequencies are considered in allele filtration **Consensus method**: Only consensus base calls and allele consensus frequencies are considered in allele filtration				