```
#!/usr/bin/perl
use warnings;
use strict;


my @samples = collate_samples();

my $amplicons = read_amplicons();

warn "Will process the following samples\n\t".join("\n\t",@samples)."\n";

foreach my $sample (@samples) {
            process_sample($sample);
}

print_allele_count_results();
print_meth_results();


sub print_meth_results {

            open
(OUT,'>','amplicon_allele_methylation_results_merged.txt') or die $!;

            # To make life easier in R we need to find the longest set of
meth calls
            my $longest_call_set = 0;

            foreach my $chr (sort keys %$amplicons) {
                        foreach my $amplicon (sort {$a->{start} <=> $b-
>{start}} @{$amplicons->{$chr}}) {
                                    foreach my $snp (sort {$a->{position}
<=> $b->{position}} @{$amplicon->{snps}}) {
                                                foreach my $sample
(@samples) {
                                                            my
@allele1_values = @{$snp->{meth1}->{$sample}};
                                                            my
@allele2_values = @{$snp->{meth2}->{$sample}};

                                                            if
(@allele1_values > $longest_call_set) {

      $longest_call_set = @allele1_values;
                                                            }

                                                            if
(@allele2_values > $longest_call_set) {

      $longest_call_set = @allele2_values;
                                                            }
                                                }
                                    }
                        }
```

```
            }

            warn "Longest call set is $longest_call_set\n";

            my @call_header;

            for (1..$longest_call_set) {
                        push @call_header,"meth_$_";
            }


            print OUT
join("\t",('Fragment','SNP','Allele','Sample',@call_header)),"\n";

            foreach my $chr (sort keys %$amplicons) {
                        foreach my $amplicon (sort {$a->{start} <=> $b-
>{start}} @{$amplicons->{$chr}}) {
                                    foreach my $snp (sort {$a->{position}
<=> $b->{position}} @{$amplicon->{snps}}) {

                                                my $fragment = $amplicon-
>{chr}.":".$amplicon->{start}.'-'.$amplicon->{end}.':'.$amplicon-
>{strand};

                                                foreach my $sample
(@samples) {
                                                            my
@allele1_values = @{$snp->{meth1}->{$sample}};
                                                            my
@allele2_values = @{$snp->{meth2}->{$sample}};
                                                            print OUT
join("\t",($fragment,$snp->{position},$snp-
>{allele1},$sample,@allele1_values)),"\n";
                                                            print OUT
join("\t",($fragment,$snp->{position},$snp-
>{allele2},$sample,@allele2_values)),"\n";
                                                }
                                    }
                        }
            }

            close OUT or die $!;

}


sub print_allele_count_results {

            open (OUT,'>','amplicon_snp_frequencies_merged.txt') or die
$!;

            print OUT
join("\t",('Fragment','SNP','Allele',@samples)),"\n";
```

```
            foreach my $chr (sort keys %$amplicons) {
                        foreach my $amplicon (sort {$a->{start} <=> $b-
>{start}} @{$amplicons->{$chr}}) {
                                    foreach my $snp (sort {$a->{position}
<=> $b->{position}} @{$amplicon->{snps}}) {

                                                my $fragment = $amplicon-
>{chr}.":".$amplicon->{start}.'-'.$amplicon->{end}.':'.$amplicon-
>{strand};

                                                my @allele1_values;
                                                foreach my $sample
(@samples) {
                                                            push
@allele1_values,$snp->{counts1}->{$sample};
                                                }

                                                print OUT
join("\t",($fragment,$snp->{position},$snp-
>{allele1},@allele1_values)),"\n";


                                                my @allele2_values;
                                                foreach my $sample
(@samples) {
                                                            push
@allele2_values,$snp->{counts2}->{$sample};
                                                }

                                                print OUT
join("\t",($fragment,$snp->{position},$snp-
>{allele2},@allele2_values)),"\n";

                                                my @alleleX_values;
                                                foreach my $sample
(@samples) {
                                                            push
@alleleX_values,$snp->{countsX}->{$sample};
                                                }

                                                print OUT
join("\t",($fragment,$snp->{position},'X',@alleleX_values)),"\n";
                                    }
                        }
            }

            close OUT or die $!;

}



sub process_sample {
            my ($sample) = @_;
```

```
            my @files = <../Data/*_${sample}_*.bam>;

            warn "Found ".scalar @files." files for $sample\n";

            foreach my $file (@files) {

                        warn "Processing $file\n";

                        open (BAM,"/bi/apps/samtools/0.1.18/samtools view
$file |") or die "Can't read $file with samtools: $!";

                        while (<BAM>) {
                                    process_entry($_,$sample);
                        }
            }
}

sub process_entry {
            my ($sam,$sample) = @_;

            my
($seq_id,$flag,$chr,$start,undef,$cigar,undef,undef,undef,$seq) =
split(/\t/,$sam);

            # Pull out the methylation call string
            my $meth_string;
            if ($sam =~ /XM:Z:([\.xhzu]+)/i) {
                        $meth_string = $1;
            }
            else {
                        die "Didn't find a methylation call string for
'$sam'";
            }

            unless (length($meth_string) == length($seq)) {
                        die "Meth string was differnt length to the
sequence\n$meth_string\n$seq\n$sam";
            }

            my $first_of_pair = 0;
            $first_of_pair = 1 if ($flag & 0x40);

            my $reversed = 0;
            $reversed = 1 if ($flag & 0x10);

            # We're not going to try to parse entries with indels at the
moment
            unless ($cigar =~ /^\d+M$/) {
#                       warn "Skipping non linear entry $cigar\n";
                        return;
            }

            my $end = $start+(length($seq)-1);
```

```
            # If we're the second of a pair then we need to flip the
sense
            # of the reveserd flag
            unless ($first_of_pair) {
                        if ($reversed) {
                                    $reversed = 0;
                        }
                        else {
                                    $reversed = 1;
                        }
            }


            my $genomic = fetch_genomic($chr,$start,$end,$reversed);


            # Find out if we're contained within an amplicon - overlap
isn't enough

            unless (exists $amplicons->{$chr}) {
#                       warn "Found no amplicons for chr '$chr'\n";
                        return;
            }
#           warn "Found ".scalar @{$amplicons->{$chr}}." amplicons on chr
$chr\n";

            foreach my $amplicon (@{$amplicons->{$chr}}) {

                        if ($start >= $amplicon->{start} and $end <=
$amplicon->{end}) {
#                                   warn "Found overlap with amplicon\n";

                                    # Check that the read is in the same
orientation as the amplicon
                                    # otherwise we're wasting our time

                                    if (($reversed and $amplicon->{strand}
eq '+') or (!$reversed and $amplicon->{strand} eq '-')) {
                                                warn "Skipping match in the
wrong direction\n";
                                                next;
                                    }


                                    # Now see whether we overlap any of the
SNP positions
                                    foreach my $snp (@{$amplicon->{snps}})
{
                                                if ($snp->{position} >=
$start and $snp->{position} <= $end) {
```

```
                                                        warn "Found
overlap with SNP at ".$snp->{position}." from start=$start and
cigar=$cigar\n";
                                                        my
$base_from_read = substr($seq,($snp->{position}-$start),1);
                                                        my
$base_from_genomic = substr($genomic,($snp->{position}-$start),1);

                                                        # If the read is
reverse we need to complement the bases we extracted
                                                        if ($reversed) {

      $base_from_read =~ tr/GATC/CTAG/;

      $base_from_genomic =~ tr/GATC/CTAG/;
                                                        }

#                                                       sleep(1);
#                                                       warn
"Read=$seq_id\tpos=$snp-
>{position}\tbase=$base_from_read\tfirst=$first_of_pair\treversed=$revers
ed\tamplicon_strand=$amplicon->{strand}\talleles=$snp->{allele1}:$snp-
>{allele2}\n";


                                                        # Our genomic
base should always match one of the possible alleles
                                                        # or something
has gone horribly wrong.

                                                        # We've since
found that there are cases where this happens, and it's because
                                                        # the UCSC
genome puts ambiguity codes into some positions, which isn't very
                                                        # helpful
                                                        unless
($base_from_genomic eq $snp->{allele1} or $base_from_genomic eq $snp-
>{allele2}) {
                                                                warn
"Posi = $chr:${start}-${end}:$reversed\n";
                                                                warn
"ID=$seq_id\n";
                                                                warn
"Geno = $genomic\n";
                                                                warn
"Read = $seq\n";
                                                                warn
"pos=$snp-
>{position}\tbase=$base_from_read\tgeno=$base_from_genomic\talleles=$snp-
>{allele1}:$snp->{allele2}\n\n";

                                                                die
"At chr=$chr pos=$snp->{position} rev=$reversed fop=$first_of_pair
```

```
genomic=$base_from_genomic wasn't $snp->{allele1} or $snp-
>{allele2}\n\n";

      #sleep(3);

      #next;
                                                                }

                                                                # Find out
whether we match allele 1, 2 or neither



                                                                # Since we're
bisulphite converted we have to do a C to T
                                                                # transformation
in both the read and the alleles.


                                                                if
($base_from_read eq 'C') {

      $base_from_read = 'T';
                                                                }

                                                                my $allele1_base
= $snp->{allele1};
                                                                if
($allele1_base eq 'C') {

      $allele1_base = 'T';
                                                                }

                                                                if
($base_from_read eq $allele1_base) {

      $snp->{counts1}->{$sample} += 1;
                                                                      if
($meth_string =~ /z/i) {

            push @{$snp->{meth1}-
>{$sample}},get_meth_percent($meth_string);
                                                                      }

      next;
                                                                }

                                                                my $allele2_base
= $snp->{allele2};
                                                                if
($allele2_base eq 'C') {

      $allele2_base = 'T';
                                                                }
```

```
                                                                    if
($base_from_read eq $allele2_base) {

      $snp->{counts2}->{$sample} += 1;
                                                                          if
($meth_string =~ /z/i) {

            push @{$snp->{meth2}-
>{$sample}},get_meth_percent($meth_string);
                                                                          }

      next;
                                                                    }

                                                                    $snp->{countsX}-
>{$sample} += 1;

                                                  }
                                        }

                                        return;
                        }
#                       else {
#                                       warn "No overlap between $start-$end
and $amplicon->{start} and $amplicon->{end}\n";
#                       }

            }

#           sleep(1);
#           warn "Found no overlaps to $chr:${start}-${end}\n";

}

sub get_meth_percent {
            my ($meth_string) = @_;

            my $meth_count = $meth_string =~ tr/Z//;
            my $unmeth_count = $meth_string =~ tr/z//;

            return ($meth_count/($meth_count+$unmeth_count))*100;


}

sub collate_samples {
            # We have multiple files for each sample so we need
    # to work out the unique set of sample names

            my %sample_names;

            my @files = <../Merged_Data/*\.bam>;
#           my @files = <../Data/*\.bam>;
```

```
            foreach my $file (@files) {
                        if ($file =~ /\/lane\d+_[GATC]+_([^_]*)_/) {
                                    next unless ($1);
                                    $sample_names{$1}++;
                        }
                        else {
                                    die "Couldn't get sample name from
$file\n";
                        }
            }

            return sort keys %sample_names;
}

sub read_amplicons {

            open (AMP,"../DMRs_SNPs.txt") or die "Can't read
DMRs_SNPs.txt: $!";

            $_ = <AMP>;

            my $amplicons;

            while (<AMP>) {
                        chomp;

                        my @sections = split(/\t/);

                        my ($chr,$start,$end) = split(/[:\-
]/,$sections[0]);

                        $chr =~ s/chr//;

                        my $amplicon = {
                                    chr => $chr,
                                    start => $start,
                                    end => $end,
                                    strand => $sections[2],
                                    snps => []
                        };

                        # Now we can add the SNPs to this amplicon

                        my $start_index = 6;

                        while (1) {
                                    my $alleles = $sections[$start_index];
                                    last unless ($alleles);
                                    my $location =
$sections[$start_index+1];

                                    # Increment the index now so we can
move on to the
```

```
                                    # next SNP at any point from now on.
                                    $start_index += 2;

                                    my ($allele1,$allele2,$check) =
split(/\//,$alleles);

                                    # Do some sanity checking on the
alleles

                                    # Move on if we have more than 2
options
                                    if ($check) {
                                              warn "Skipping SNP where
there are more than 2 alleles\n";
                                              next;
                                    }

                                    # We're currently avoiding multi base
alleles or
                                    # alleles containing a deletion since
both of these
                                    # will cause complications.

                                    unless ($allele1 =~ /^[GATC]$/) {
                                              warn "Skipping SNP where
allele 1 is '$allele1'\n";
                                              next;
                                    }

                                    unless ($allele2 =~ /^[GATC]$/) {
                                              warn "Skipping SNP where
allele 2 is '$allele2'\n";
                                              next;
                                    }


                                    if ($allele1 eq 'C' and $allele2 eq 'T'
or $allele1 eq 'T' and $allele2 eq 'C') {
                                              warn "Skipping C to T SNP
which we can't distinguish\n";
                                              next;
                                    }


                                    my (undef,$pos) = split(/[:\-
]/,$location);

                                    my $snp = {position => $pos,
                                                                        allele1
=> $allele1,
                                                                        allele2
=> $allele2,
                                                                        counts1
=> {},
```

```
                                                                        counts2
=> {},
                                                                        meth1 =>
{},
                                                                        meth2 =>
{},
                                                                        countsX
=> {}, # Seqs other than expected
                                        };

                                        foreach my $sample (@samples) {
                                                $snp->{counts1}->{$sample}
= 0;
                                                $snp->{counts2}->{$sample}
= 0;
                                                $snp->{countsX}->{$sample}
= 0;
                                                $snp->{meth1}->{$sample} =
[];
                                                $snp->{meth2}->{$sample} =
[];
                                        }

                                        push @{$amplicon->{snps}},$snp;

                                }

                                push @{$amplicons->{$chr}},$amplicon;
                        }

                        return $amplicons;

}


{

                        my %chrs;

                        sub fetch_genomic {
                                my ($chr,$start,$end,$reverse) = @_;

                                unless (exists $chrs{$chr}) {

                                        warn "Loading chr $chr\n";

                                        open
(CHR,"/bi/scratch/Genomes/Human/GRCh37/Homo_sapiens.GRCh37.55.dna.chromos
ome.${chr}.fa") or die "Can't open fasta for chr $chr: $!";

                                        $_ = <CHR>;
                                        my $seq;

                                        while (<CHR>) {
```

```
                                        s/[\r\n]//g;

                                        $seq .= $_;
                                }

                                $chrs{$chr} = $seq;
                        }


                        my $seq = substr($chrs{$chr},$start-1,($end-
$start)+1);

#                       if ($reverse) {
#                               $seq = reverse($seq);
#                               $seq =~ tr/GATC/CTAG/;
#                       }

                        return ($seq);
                }

}
```