**Usage: Confirm validity of reference sequences vs vcf files**

**Author: Temi**

**Date: Tues Jan 24 2023**

```
import os
import numpy as np
import cyvcf2
import kipoiseq
import pandas as pd
```

```
working_dir = '/grand/covid-ct/imlab/users/temi/projects/TFXcan/scripts/'
os.chdir(working_dir)
```

```
reference_genome = '/grand/covid-ct/imlab/data/hg_sequences/hg38/Homo_sapiens_assembly38.f
vcf_file = '../genotypes/prj6_genotypes/merged_phased_SNPs.vcf.gz'
```

Use kipoiseq to read fasta file

```
class FastaStringExtractor:
    def __init__(self, fasta_file):
        import pyfaidx

        self.fasta = pyfaidx.Fasta(fasta_file)
        self._chromosome_sizes = {k: len(v) for k, v in self.fasta.items()}

    def extract(self, interval, **kwargs) -> str:
        # Truncate interval if it extends beyond the chromosome lengths.

        import kipoiseq
        chromosome_length = self._chromosome_sizes[interval.chrom]
        trimmed_interval = kipoiseq.Interval(interval.chrom,
                                            max(interval.start, 0),
                                            min(interval.end, chromosome_length),
                                            )
        # pyfaidx wants a 1-based interval
        sequence = str(self.fasta.get_seq(trimmed_interval.chrom,
                                          trimmed_interval.start + 1,
                                          trimmed_interval.stop).seq).upper()
        # Fill truncated values with N's.
        pad_upstream = 'N' * max(-interval.start, 0)
        pad_downstream = 'N' * max(interval.end - chromosome_length, 0)
        return pad_upstream + sequence + pad_downstream

    def close(self):
        return self.fasta.close()
```

```
fasta_extractor = FastaStringExtractor(reference_genome)
```

A short snippet

```
region_chr = 'chr1'
region_start = 100001
region_end = 100009
SEQUENCE_LENGTH=len(range(region_start, region_end))
```

```
reg_interval = kipoiseq.Interval(region_chr, region_start, region_end).resize(SEQUENCE_LEN
reference_region = fasta_extractor.extract(interval=reg_interval, anchor=[])
reference_region
```

```
'CTAAGCAC'
```

### The experiment

Use cyvcf to extract vcf

There are currently about 11 million variants in this vcf file

For this purpose of this notebook, I will use the top 1 million variants (for memory sake)

```
stop_at = 11000000
```

```
all_variants = []

for i, variant in enumerate(cyvcf2.cyvcf2.VCF(vcf_file)):
    if i > stop_at:
        break
    else:
        out = []
        out.extend([variant.CHROM, variant.start, variant.end, variant.REF, variant.ALT])
        all_variants.append(out)
```

```
[W::hts_idx_load3] The index file is older than the data file: ../genotypes/prj6_genotypes/m
```

Here is what the output looks like ( I think it went over by one but it does not matter)

```
all_variants[0:3] ; len(all_variants)
```

```
11000001
```

Extract the same regions from the reference genome

```
reference_variants_positions = [[v[0], v[1], v[2]] for v in all_variants]
reference_variants_positions[1:5]
```

```
[['chr1', 10247, 10248],
 ['chr1', 10462, 10463],
 ['chr1', 10491, 10492],
 ['chr1', 13272, 13273]]
```

```
len(reference_variants_positions)
```

```
11000001
```

```
reference_variants = []
for vpos in reference_variants_positions:
    slength = len(range(vpos[1], vpos[2]))
    reg_interval = kipoiseq.Interval(vpos[0], vpos[1], vpos[2]).resize(slength)
    reference_region = fasta_extractor.extract(interval=reg_interval, anchor=[])
    reference_variants.append(reference_region)
```

Convert to a string

```
reference_variants = ''.join(reference_variants)
```

Convert the vcf variants to a string too

```
vcf_ref_variants = ''.join([v[3] for v in all_variants])
```

Compare the reference variants with the vcf variants

```
reference_variants == vcf_ref_variants
```

```
True
```

```
len(reference_variants), len(vcf_ref_variants)
```

```
(11000001, 11000001)
```

They are the same