Kmer pipeline

Pipeline for performing nucleotide and protein kmer GWAS analyses using Nextflow.

Sarah G Earle and Daniel J Wilson, October 2022

University of Oxford Big Data Institute

Implementing methods described in

Genome-wide association studies of global *Mycobacterium tuberculosis* resistance to thirteen antimicrobials in 10,228 genomes The CRyPTIC Consortium (2022) *PLOS Biology* 20: e3001755.

Identifying lineage effects when controlling for population structure improves power in bacterial association studies. Earle, S. G., Wu, C.-H., Charlesworth, J., Stoesser, N., Gordon, N. C., Walker, T. M., Spencer, C. C. A., Iqbal, Z., Clifton, D. A., Hopkins, K. L., Woodford, N., Smith, E. G., Ismail, N., Llewelyn, M. J., Peto, T. E., Crook, D. W., McVean, G., Walker, A. S. and D. J. Wilson (2016) *Nature Microbiology* 1: 16041 (preprint)

Input

Phenotypes	Binary or continuous phenotypes. If any phenotypes are NA, the samples with NA phenotypes will be ignored in the LMM and when determining the minor allele counts/frequencies but will be included in the other pattern files.
Genotypes	Kmer counting input is assembly contigs, not sequencing reads. Kmers are counted as present if seen once in a genome.

License

Contents

Installation

OUTPUT

Docker container **dannywilson/kmer_pipeline:2022-10-26** or Singularity container **kmer_pipeline_2022-10-26.sif** and Nextflow pipeline **kmer_pipeline.nf**

1. Singularity container: pull from DockerHub (*recommended*)

Prerequisites: Singularity installation, up to 10 GiB storage, 1 CPU.

Open a terminal session where Singularity is available, e.g. via ssh.

Build a Singularity image by pulling the DockerHub image. Create a variable referencing the container.

```
singularity pull -F docker://dannywilson/kmer_pipeline:2022-10-26
```

Copy the Nextflow pipeline to a local file

```
singularity exec --containall --cleanenv kmer_pipeline_2022-10-26.sif cat
/usr/local/bin/kmer_pipeline.nf > ./kmer_pipeline.nf
```

2. Docker container: pull from DockerHub (*recommended*)

Prerequisites: Docker installation, up to 10 GiB storage, 1 CPU.

Open a terminal session where Docker is available, e.g. via ssh.

Pull the Docker image from DockerHub

```
docker pull dannywilson/kmer_pipeline:2022-10-26
```

Copy the Nextflow pipeline to a local file

```
docker run --rm dannywilson/kmer_pipeline:2022-10-26 cat
/usr/local/bin/kmer_pipeline.nf > ./kmer_pipeline.nf
```

3. Docker container: manual build (*not recommended)*

Prerequisites: Docker and Git installations, up to 10 GiB storage, 1 CPU.

Open a terminal session where Docker and Git are available, e.g. via ssh.

Create a local directory and clone a specific release of the repository

```
git clone --depth 1 --branch 2022-10-26 https://github.com/danny-
wilson/kmer_pipeline.git
```

Build the Docker image from the dockerfile

```
cd kmer_pipeline
docker build -t dannywilson/kmer_pipeline:2022-10-26 .
```

Copy the Nextflow pipeline to a local file

```
docker run --rm dannywilson/kmer_pipeline:2022-10-26 cat
/usr/local/bin/kmer_pipeline.nf > ./kmer_pipeline.nf
```

OUTPUT

Executes the kmer pipeline end-to-end.
One file ending
.report.html
One directory ending
_kmergenealign_figures
Contains a summary of the GWAS results readable in a web browser, including Manhattan plots, QQ plots, tables of significant regions and kmers.

See previous section for installation of the Singularity container.

1. Nextflow inside Singularity (*quick start*)

Prerequisites: Singularity, kmer_pipeline Singularity container

If you have Nextflow installed, it is recommended to start with Nextflow outside Singularity (next example). That makes it simpler to understand the configuration files, in which file paths are specified relative to the user file system, rather than the container file system as here.

Begin by defining the location, on the user file system, of the Singularity container. E.g.

```
CONTAINER=/users/username/kmer_pipeline/kmer_pipeline_2022-10-26.sif
```

Replace the above with the full path and filename of the Singularity container on your system.

Next locate the mountpoint: the directory to share with the container that allows it read/write access. This directory must contain, directly or through its subdirectories, all the input files, and the location for the output files in your analysis. E.g.

```
MNT_DIR=/users/username/kmer_pipeline
```

Replace the above with the full path of the desired mountpoint on your system.

Within the container, `MNT_DIR` will be visible as `/home/jovyan`. Within `MNT_DIR` create a subdirectory called `tb20` to act as the base directory for the example analysis; this is the `base_dir` that is specified in the example `nextflow.config`.

```
BASE_DIR=$MNT_DIR/tb20
mkdir -p $BASE_DIR
```

Open an interactive bash shell within the container.

```
singularity exec --containall --cleanenv --home $MNT_DIR:/home/jovyan
$CONTAINER bash
```

Now within the container, copy the example configuration file to the base directory, renaming it `nextflow.config`

```
cp /usr/share/kmer_pipeline/example/baremetal.nextflow.config
~/tb20/nextflow.config
```

Navigate to the base directory and launch the analysis; Nextflow automatically reads the `nextflow.config` file in the current directory.

```
cd tb20 && kmer_pipeline.nf
```

In testing, this took about 15 minutes with two CPUs or 25 minutes with one CPU. When the analysis is done, type `exit` to close the interactive container. The output files will be in `$BASE_DIR` on the user file system.

To view the results, use a web browser to open the `*.report.html` file in `$BASE_DIR/kmergwas` (i.e. the location on the user file system implied by `analysis_dir` in `nextflow.config`). If downloading the results from a remote server, make sure to download all files matching `*report*` in `$BASE_DIR/kmergwas` plus the subdirectory `*_kmergenealign_figures`.

Adapting this example for your data: In this example, the entire analysis is run *within* the container. Output files are mutually visible because `$MNT_DIR` on the user file system is *mounted* to `/home/jovyan` on the container file system. For your own analysis, you need to:

- Modify `nextflow.config` as required, for example varying file locations and setting `maxp` to the number of CPUs available. Make sure `nextflow.config` is situated in the base directory, from which you will launch Nextflow.

- Modify the `id_file` specified in `nextflow.config` to contain your own sample IDs, genome assembly paths, and phenotypes.

- **NB: when running kmer_pipeline *inside* a container,** file locations in `nextflow.config`, and the `paths` column in `id_file` must all contain the full absolute paths for the location of genome assemblies on the *container file system* because the container cannot directly see the user file system. For example, if the assembly `/usr/username/kmer_pipeline/contigs/assembly1.fa.gz` is the full absolute path on the *user file system*, and if `MNT_DIR=/usr/username/kmer_pipeline` then the full absolute path on the *container file system* will be `/home/jovyan/contigs/assembly1.fa.gz`. Likewise, if `nextflow.config`, specifies `base_dir = "/home/jovyan/tb20"`, that implies the base directory on the user file system will be `$MNT_DIR/tb20`.

2. Nextflow outside Singularity (*recommended on bare metal machines*)

Prerequisites: Nextflow, Singularity, kmer_pipeline Singularity container

In this approach, Singularity is called by Nextflow, and the contents of `nextflow.config` and `id_file` refer to files in the user file system, which is simpler. However, Nextflow must now be installed on the user machine. This approach is limited by the number of CPUs and the amount of RAM available on the user machine. For larger scale analysis, Nextflow on a cluster (next example) is recommended.

Begin by defining the location, on the user file system, of the Singularity container. E.g.

```
CONTAINER=/users/username/kmer_pipeline/kmer_pipeline_2022-10-26.sif
```

Replace the above with the full path and filename of the Singularity container on your system.

Next locate the base directory for the analysis. Nextflow will automatically make this the mountpoint: the directory to share with the container that allows it read/write access. This directory must contain, directly or through its subdirectories, all the input files, and the location for the output files in your analysis. E.g.

```
BASE_DIR=/users/username/kmer_pipeline
```

Replace the above with the full path of the desired base directory/mountpoint on your system.

If it does not already exist, create the base directory and a subdirectory called `tb20` to contain the example data.

```
mkdir -p $BASE_DIR/tb20
```

Next extract the example files to `$BASE_DIR/tb20` in the user file system.

```
cd $BASE_DIR/tb20 && singularity exec --containall --cleanenv $CONTAINER bash
-c 'cd /usr/share/kmer_pipeline/example && tar -c .' | tar -x
```

Now edit the example configuration file to point to the correct locations on the user file system. First edit the location of the container path:

```
sed "s,YOUR_CONTAINER_PATH_HERE/kmer_pipeline_2022-10-26.sif,$CONTAINER,g"
singularity.nextflow.config > $BASE_DIR/nextflow.config
```

Next substitute the correct base directory in the configuration file:

```
sed -i "s,YOUR_PATH_HERE,$BASE_DIR,g" $BASE_DIR/nextflow.config
```

The `id_file` must also be updated to give the location of the example genome assemblies on the user file system:

```
sed -i "s,/usr/share/kmer_pipeline/example/,$BASE_DIR/tb20/,g"
$BASE_DIR/tb20/id_file.txt
```

Extract the kmer_pipeline Nextflow script to the base directory:

```
singularity exec --containall --cleanenv $CONTAINER cat
/usr/local/bin/kmer_pipeline.nf > $BASE_DIR/kmer_pipeline.nf
```

Navigate to the base directory and launch the analysis; Nextflow automatically reads the `nextflow.config` file in the current directory.

```
cd $BASE_DIR && nextflow kmer_pipeline.nf
```

In testing, this took about 15 minutes with two CPUs or 25 minutes with one CPU. The output files will be in `$BASE_DIR` on the user file system.

To view the results, use a web browser to open the `*.report.html` file in `$BASE_DIR/tb20/kmergwas` (i.e. the location on the user file system specified by `analysis_dir` in `nextflow.config`). If downloading the results from a remote server, make sure to download all files matching `*report*` in `$BASE_DIR/tb20/kmergwas` plus the subdirectory `*_kmergenealign_figures`.

Adapting this example for your data: In this example, the user interacts with Nextflow directly, which handles calls to the container. The locations of input and output files are clear because in `nextflow.config` and `id_file` they are specified in full absolute paths on the *user file system*. For your own analysis, you need to:

- Modify `nextflow.config` as required, for example varying file locations and setting `maxp` to the number of CPUs available. Make sure `nextflow.config` is situated in the base directory, from which you will launch Nextflow.

- Modify the `id_file` specified in `nextflow.config` to contain your own sample IDs, genome assembly paths, and phenotypes.

Refer to the later reference for an explanation of all parameters in `nextflow.config`.

3. Nextflow on a cluster using Singularity (*recommended at scale*)

Prerequisites: Cluster environment supported by Nextflow (e.g. Sun Grid Engine, SLURM), Nextflow, Singularity, kmer_pipeline Singularity container

This approach is recommended for analysis at scale (e.g. hundreds or more genomes). However, for troubleshooting, the simpler approach of the previous example is suggested (Nextflow outside Singularity), starting with ensuring that the example analysis can be run. Like in the previous example, all file names are specified with full absolute paths on the user file system.

Begin by defining the location, on the user file system, of the Singularity container. E.g.

```
CONTAINER=/users/username/kmer_pipeline/kmer_pipeline_2022-10-26.sif
```

Replace the above with the full path and filename of the Singularity container on your system.

Next locate the base directory for the analysis. Nextflow will automatically make this the mountpoint: the directory to share with the container that allows it read/write access. This directory must contain, directly or through its subdirectories, all the input files, and the location for the output files in your analysis. E.g.

```
BASE_DIR=/users/username/kmer_pipeline
```

Replace the above with the full path of the desired base directory/mountpoint on your system.

If it does not already exist, create the base directory and a subdirectory called `tb20` to contain the example data.

```
mkdir -p $BASE_DIR/tb20
```

Next extract the example files to `$BASE_DIR/tb20` in the user file system.

```
cd $BASE_DIR/tb20 && singularity exec --containall --cleanenv $CONTAINER bash
-c 'cd /usr/share/kmer_pipeline/example && tar -c .' | tar -x
```

Now edit the example configuration file to point to the correct locations on the user file system. First edit the location of the container path:

```
sed "s,YOUR_CONTAINER_PATH_HERE/kmer_pipeline_2022-10-26.sif,$CONTAINER,g"
sge.nextflow.config > $BASE_DIR/nextflow.config
```

Next substitute the correct base directory in the configuration file:

```
sed -i "s,YOUR_PATH_HERE,$BASE_DIR,g" $BASE_DIR/nextflow.config
```

Note the addition of a `process` section in `nextflow.config` which was absent from the previous example – this is the only difference in configuration between the two examples. The `process` section contains two parameters, `executor` and `queue`. **These must be customized for your system, *particularly***

the queue name. Specify the type of cluster, e.g. "sge" or "slurm" and the queue (or partition) name; refer to the Nextflow documentation for further details.

EXECUTOR="sge"
QUEUE="short.qc"

Another difference compared to the previous example is that `maxp` is set to `30`, the number of genomes. If you have more than 30 CPUs available, you could increase `maxp`, although the gain on the example data is likely to be marginal. Note that not all steps in kmer_pipeline can utilize all available CPUs. For example, the degree of parallelization in some steps is determined by the number of genomes.

Now substitute your customized values into `nextflow.config`:

```
sed -i "s,sge,$EXECUTOR,g" $BASE_DIR/nextflow.config
sed -i "s,short.qc,$QUEUE,g" $BASE_DIR/nextflow.config
```

The `id_file` must also be updated to give location of the example genome assemblies on the user file system:

```
sed -i "s,/usr/share/kmer_pipeline/example/,$BASE_DIR/tb20/,g"
$BASE_DIR/tb20/id_file.txt
```

Extract the kmer_pipeline Nextflow script to the base directory:

```
singularity exec --containall --cleanenv $CONTAINER cat
/usr/local/bin/kmer_pipeline.nf > $BASE_DIR/kmer_pipeline.nf
```

Navigate to the base directory and launch the analysis; Nextflow automatically reads the `nextflow.config` file in the current directory.

```
cd $BASE_DIR && nextflow kmer_pipeline.nf
```

In testing, this took about 10 minutes with 30 CPUs, but the run time can be strongly influenced by time spent queuing on the cluster. The output files will be in `$BASE_DIR` on the user file system.

To view the results, use a web browser to open the `*.report.html` file in `$BASE_DIR/tb20/kmergwas` (i.e. the location on the user file system specified by `analysis_dir` in `nextflow.config`). If downloading the results from a remote server, make sure to download all files matching `*report*` in `$BASE_DIR/tb20/kmergwas` plus the subdirectory `*_kmergenealign_figures`.

Adapting this example for your data: In this example, the user interacts with Nextflow directly, which handles calls to the container via the cluster management software. The locations of input and output files are clear because in `nextflow.config` and `id_file` they are specified in full absolute paths on the *user file system*. For your own analysis, you need to:

- Modify `nextflow.config` as required, for example varying file locations, setting `maxp` to the number of CPUs available, and ensuring the `queue` name is set correctly. Make sure `nextflow.config` is situated in the base directory, from which you will launch Nextflow.
- Modify the `id_file` specified in `nextflow.config` to contain your own sample IDs, genome assembly paths, and phenotypes.

Refer to the later reference for an explanation of all parameters in `nextflow.config`.

OUTPUT

Executes the kmer pipeline end-to-end.
One file ending
.report.html
One directory ending
_kmergenealign_figures
Contains a summary of the GWAS results readable in a web browser, including Manhattan plots, QQ plots, tables of significant regions and kmers.

See earlier section for installation of the Docker container.

1. Docker inside Singularity (*quick start*)

Prerequisites: Docker, kmer_pipeline Docker image

If you have Nextflow installed, it is recommended to start with Nextflow outside Docker (next example). That makes it simpler to understand the configuration files, in which file paths are specified relative to the user file system, rather than the container file system as here.

Begin by defining the name of the Docker image downloaded earlier. E.g.

```
CONTAINER="dannywilson/kmer_pipeline:2022-10-26"
```

Replace the above with the name of the Docker image on your system, if different.

Next locate the mountpoint: the directory to share with the container that allows it read/write access. This directory must contain, directly or through its subdirectories, all the input files, and the location for the output files in your analysis. E.g.

```
MNT_DIR=/users/username/kmer_pipeline
```

Replace the above with the full path of the desired mountpoint on your system.

Within the container, `MNT_DIR` will be visible as `/home/jovyan`. Within `MNT_DIR` create a subdirectory called `tb20` to act as the base directory for the example analysis; this is the `base_dir` that is specified in the example `nextflow.config`.

```
BASE_DIR=$MNT_DIR/tb20
mkdir -p $BASE_DIR
```

Open an interactive bash shell within the container.

```
docker run -it --rm -v $MNT_DIR:/home/jovyan $CONTAINER bash
```

Now within the container, copy the example configuration file to the base directory, renaming it `nextflow.config`

```
cp /usr/share/kmer_pipeline/example/baremetal.nextflow.config
~/tb20/nextflow.config
```

Navigate to the base directory and launch the analysis; Nextflow automatically reads the `nextflow.config` file in the current directory.

```
cd tb20 && kmer_pipeline.nf
```

In testing, this took about 15 minutes with two CPUs or 25 minutes with one CPU. When the analysis is done, type `exit` to close the interactive container. The output files will be in `$BASE_DIR` on the user file system.

To view the results, use a web browser to open the `*.report.html` file in `$BASE_DIR/kmergwas` (i.e. the location on the user file system implied by `analysis_dir` in `nextflow.config`). If downloading the results from a remote server, make sure to download all files matching `*report*` in `$BASE_DIR/kmergwas` plus the subdirectory `*_kmergenealign_figures`.

Adapting this example for your data: In this example, the entire analysis is run *within* the container. Output files are mutually visible because `$MNT_DIR` on the user file system is *mounted* to `/home/jovyan` on the container file system. For your own analysis, you need to:

- Modify `nextflow.config` as required, for example varying file locations and setting `maxp` to the number of CPUs available. Make sure `nextflow.config` is situated in the base directory, from which you will launch Nextflow.

- Modify the `id_file` specified in `nextflow.config` to contain your own sample IDs, genome assembly paths, and phenotypes.

- **NB: when running kmer_pipeline *inside* a container,** file locations in `nextflow.config`, and the `paths` column in `id_file` must all contain the full absolute paths for the location of genome assemblies on the *container file system* because the container cannot directly see the user file system. For example, if the assembly `/usr/username/kmer_pipeline/contigs/assembly1.fa.gz` is the full absolute path on the *user file system*, and if `MNT_DIR=/usr/username/kmer_pipeline` then the full absolute path on the *container file system* will be `/home/jovyan/contigs/assembly1.fa.gz`. Likewise, if `nextflow.config`, specifies `base_dir = "/home/jovyan/tb20"`, that implies the base directory on the user file system will be `$MNT_DIR/tb20`.

2. Nextflow outside Docker (*recommended on bare metal machines*)

Prerequisites: Nextflow, Docker, kmer_pipeline Docker image

In this approach, Docker is called by Nextflow, and the contents of `nextflow.config` and `id_file` refer to files in the user file system, which is simpler. However, Nextflow must now be installed on the user machine. This approach is limited by the number of CPUs and the amount of RAM available on the user machine. For larger scale analysis, Nextflow on a cluster (next example) is recommended.

Begin by defining the name of the Docker image downloaded earlier. E.g.

```
CONTAINER="dannywilson/kmer_pipeline:2022-10-26"
```

Replace the above with the name of the Docker image on your system, if different.

Next locate the base directory for the analysis. Nextflow will automatically make this the mountpoint: the directory to share with the container that allows it read/write access. This directory must contain, directly or through its subdirectories, all the input files, and the location for the output files in your analysis. E.g.

```
BASE_DIR=/users/username/kmer_pipeline
```

Replace the above with the full path of the desired base directory/mountpoint on your system.

If it does not already exist, create the base directory and a subdirectory called `tb20` to contain the example data.

```
mkdir -p $BASE_DIR/tb20
```

Next extract the example files to `$BASE_DIR/tb20` in the user file system.

```
cd $BASE_DIR/tb20 && docker run --rm $CONTAINER bash -c 'cd
/usr/share/kmer_pipeline/example && tar -c .' | tar -x
```

Now edit the example configuration file to point to the correct locations on the user file system. First edit the location of the container path:

```
sed "s,dannywilson/kmer_pipeline:2022-10-26,$CONTAINER,g"
docker.nextflow.config > $BASE_DIR/nextflow.config
```

Next substitute the correct base directory in the configuration file:

```
sed -i "s,YOUR_PATH_HERE,$BASE_DIR,g" $BASE_DIR/nextflow.config
```

The `id_file` must also be updated to give location of the example genome assemblies on the user file system:

```
sed -i "s,/usr/share/kmer_pipeline/example/,$BASE_DIR/tb20/,g"
$BASE_DIR/tb20/id_file.txt
```

Extract the kmer_pipeline Nextflow script to the base directory:

```
docker run --rm $CONTAINER cat /usr/local/bin/kmer_pipeline.nf >
$BASE_DIR/kmer_pipeline.nf
```

Navigate to the base directory and launch the analysis; Nextflow automatically reads the `nextflow.config` file in the current directory.

```
cd $BASE_DIR && nextflow kmer_pipeline.nf
```

In testing, this took about 15 minutes with two CPUs or 25 minutes with one CPU. The output files will be in `$BASE_DIR` on the user file system.

To view the results, use a web browser to open the `*.report.html` file in `$BASE_DIR/tb20/kmergwas` (i.e. the location on the user file system specified by `analysis_dir` in `nextflow.config`). If downloading the results from a remote server, make sure to download all files matching `*report*` in `$BASE_DIR/tb20/kmergwas` plus the subdirectory `*_kmergenealign_figures`.

Adapting this example for your data: In this example, the user interacts with Nextflow directly, which handles calls to the container. The locations of input and output files are clear because in `nextflow.config` and `id_file` they are specified in full absolute paths on the *user file system*. For your own analysis, you need to:

- Modify `nextflow.config` as required, for example varying file locations and setting `maxp` to the number of CPUs available. Make sure `nextflow.config` is situated in the base directory, from which you will launch Nextflow.

- Modify the `id_file` specified in `nextflow.config` to contain your own sample IDs, genome assembly paths, and phenotypes.

Refer to the later reference for an explanation of all parameters in `nextflow.config`.

3. Nextflow on a cluster using Docker (*this or Singularity recommended at scale*)

Prerequisites: Cluster environment supported by Nextflow (e.g. Sun Grid Engine, SLURM), Nextflow, Docker, kmer_pipeline Docker image

NB: often research cluster administrators will not install Docker for security reasons, which is why a guide to Singularity is also included. The cluster approach is recommended for analysis at scale (e.g. hundreds or more genomes). However, for troubleshooting, the simpler approach of the previous example is suggested (Nextflow outside Docker or Singularity), starting with ensuring that the example analysis can be run. Like in the previous example, all file names are specified with full absolute paths on the user file system.

Begin by defining the name of the Docker image downloaded earlier. E.g.

```
CONTAINER="dannywilson/kmer_pipeline:2022-10-26"
```

Replace the above with the name of the Docker image on your system, if different.

Next locate the base directory for the analysis. Nextflow will automatically make this the mountpoint: the directory to share with the container that allows it read/write access. This directory must contain, directly or through its subdirectories, all the input files, and the location for the output files in your analysis. E.g.

```
BASE_DIR=/users/username/kmer_pipeline
```

Replace the above with the full path of the desired base directory/mountpoint on your system.

If it does not already exist, create the base directory and a subdirectory called `tb20` to contain the example data.

```
mkdir -p $BASE_DIR/tb20
```

Next extract the example files to `$BASE_DIR/tb20` in the user file system.

```
cd $BASE_DIR/tb20 && docker run --rm $CONTAINER bash -c 'cd
/usr/share/kmer_pipeline/example && tar -c .' | tar -x
```

Now edit the example configuration file to point to the correct locations on the user file system. First edit the location of the container path:

```
sed "s,dannywilson/kmer_pipeline:2022-10-26,$CONTAINER,g" sge.nextflow.config
> $BASE_DIR/nextflow.config
```

Next substitute the correct base directory in the configuration file:

```
sed -i "s,YOUR_PATH_HERE,$BASE_DIR,g" $BASE_DIR/nextflow.config
```

Since the template was written with Singularity in mind, it is also necessary to edit the container type:

```
sed -i "s,singularity,docker,g" $BASE_DIR/nextflow.config
```

Note the addition of a `process` section in `nextflow.config` which was absent from the previous example – this is the only difference in configuration between the two examples. The `process` section contains two parameters, `executor` and `queue`. **These must be customized for your system, *particularly the queue name.*** Specify the type of cluster, e.g. "sge" or "slurm" and the queue (or partition) name; refer to the Nextflow documentation for further details.

EXECUTOR="sge"
QUEUE="short.qc"

Another difference compared to the previous example is that `maxp` is set to `30`, the number of genomes. If you have more than 30 CPUs available, you could increase `maxp`, although the gain on the example data is likely to be marginal. Note that not all steps in kmer_pipeline can utilize all available CPUs. For example, the degree of parallelization in some steps is determined by the number of genomes.

Now substitute your customized values into `nextflow.config`:

```
sed -i "s,sge,$EXECUTOR,g" $BASE_DIR/nextflow.config
sed -i "s,short.qc,$QUEUE,g" $BASE_DIR/nextflow.config
```

The `id_file` must also be updated to give the location of the example genome assemblies on the user file system:

```
sed -i "s,/usr/share/kmer_pipeline/example/,$BASE_DIR/tb20/,g"
$BASE_DIR/tb20/id_file.txt
```

Extract the kmer_pipeline Nextflow script to the base directory:

```
docker run --rm $CONTAINER cat /usr/local/bin/kmer_pipeline.nf >
$BASE_DIR/kmer_pipeline.nf
```

Navigate to the base directory and launch the analysis; Nextflow automatically reads the `nextflow.config` file in the current directory.

```
cd $BASE_DIR && nextflow kmer_pipeline.nf
```

In testing, the Singularity version took about 10 minutes with 30 CPUs, but the run time can be strongly influenced by time spent queuing on the cluster. The output files will be in `$BASE_DIR` on the user file system.

To view the results, use a web browser to open the `*.report.html` file in `$BASE_DIR/tb20/kmergwas` (i.e. the location on the user file system specified by `analysis_dir` in `nextflow.config`). If downloading the results from a remote server, make sure to download all files matching `*report*` in `$BASE_DIR/tb20/kmergwas` plus the subdirectory `*_kmergenealign_figures`.

Adapting this example for your data: In this example, the user interacts with Nextflow directly, which handles calls to the container via the cluster management software. The locations of input and output files are clear because in `nextflow.config` and `id_file` they are specified in full absolute paths on the *user file system*. For your own analysis, you need to:

- Modify `nextflow.config` as required, for example varying file locations, setting `maxp` to the number of CPUs available, and ensuring the `queue` name is set correctly. Make sure `nextflow.config` is situated in the base directory, from which you will launch Nextflow.

- Modify the `id_file` specified in `nextflow.config` to contain your own sample IDs, genome assembly paths, and phenotypes.

Refer to the later reference for an explanation of all parameters in `nextflow.config`.

Example output: Candidate gene analysis of rifampicin resistance in *M. tuberculosis*

The commands in the previous two sections run an example analysis of $\log_2$ rifampicin minimum inhibitory concentration (MIC) in 30 *Mycobacterium tuberculosis* genomes, focusing on 20 candidate genes comprising the known causal gene, *rpoB*, and 19 non-causal genes: PE_PGRS52, Rv0115a, Rv0374c, Rv0481c, Rv0537c, Rv2060, Rv2819c, Rv3060c, Rv3352c, Rv3551, Rv3592, Rv3831, *drrB*, *lpqW*, *ltp4*, *moaA2*, *murF*, *uvrA* and *vapB16*. The example data is an extract from CRYPTIC (2022) *PLOS Biology* 20: e3001755.

A candidate gene approach is not recommended; it was produced to facilitate a modest-sized, quick-to-run example dataset.

Successful execution of the pipeline produces Nextflow output like this:

```
executor >  sge (164)
[bc/17d037] process > countkmers (18)                   [100%] 30 of 30 ✓
[0a/b217fc] process > createfullkmerlist (15)           [100%] 15 of 15 ✓
[d5/1bf4b2] process > stringlist2patternandkinship (2)  [100%] 30 of 30 ✓
[00/e7ac84] process > rungemma (7)                      [100%] 30 of 30 ✓
[1e/fa44e4] process > kmercontigalign (11)              [100%] 30 of 30 ✓
[59/53cca5] process > kmercontigalignmerge (5)          [100%] 6 of 6 ✓
[9c/b3eea3] process > plotManhattan                     [100%] 1 of 1 ✓
[22/0634d4] process > genReport                         [100%] 1 of 1 ✓
[37/bb868d] process > genGeneReport (13)                [100%] 20 of 20 ✓
[97/73e23f] process > genUnmappedReport                 [100%] 1 of 1 ✓
Completed at: 27-Oct-2022 09:20:18
Duration    : 9m 8s
CPU hours   : 0.7
Succeeded   : 164
```

Within the base directory is a subdirectory called `kmergwas`, containing the HTML report file `tb20_nucleotide31.report.html`. This file references other report files named `*report*` and the contents of the subdirectory `nucleotidekmer31_kmergenealign_figures`.

A minimal archive of an analysis would save the container, `nextflow.config`, `id_file`, the input genomes, the `*report*` files and the `*_kmergenealign_figures` subdirectory.

For convenience, you may wish to download the `*report*` files and `*_kmergenealign_figures` subdirectory to view the report in a web browser on your local machine.

The file `*.report.html` is the index of the report. It contains a summary of the analysis parameters, and sections reporting on heritability, significance threshold, most significant regions, Manhattan plot and QQ plots. The example report is recapitulated below:

Kmer GWAS report

```
Prefix: tb20; KmerType: nucleotide; K: 31; ReferenceGenome: NC_000962.3; MAF: 0.01;
MinCount: 1; AlignIdent: 90; ReportTimeStamp: Wed Oct 26 10:03:44 2022.
```

Heritability

The sample heritability (proportion of variance explained) under the null linear mixed model (LMM) was 0.917 with a standard error of 0.102, which implies a 95% confidence interval of (0.718, 1.00).

Significance threshold

A total of 25015 distinct kmers were observed, of which there were 183 unique phylopatterns (patterns of presence or absence) across the sample. After filtering any individuals lacking phenotype information, and applying a minor allele frequency (MAF) threshold of 0.01, there were 182 unique phylopatterns to be tested. Assuming a familywise error rate of 5%, this implied a Bonferroni-corrected *p*-value threshold of 0.000275, or $10^{-3.56}$.

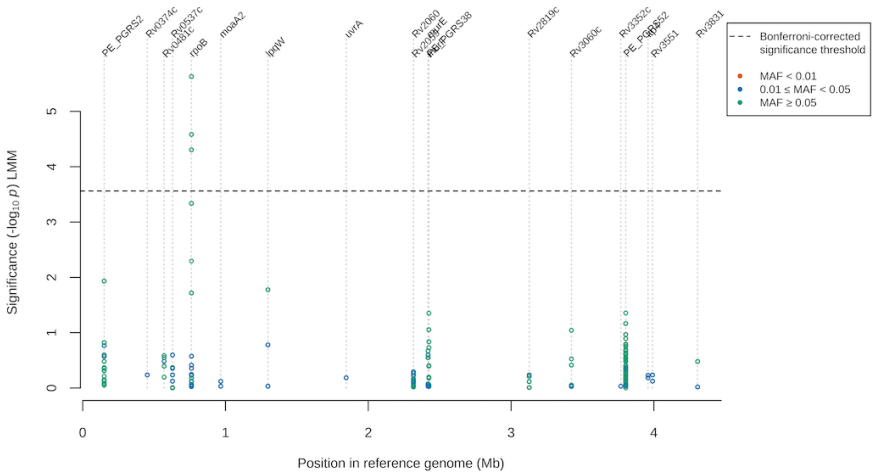
Most significant regions

The 20 most significant genes or intergenic regions are summarized in the Table below. Of those, 1 were genome-wide significant. The gene or (if an intergenic region) flanking genes are named for each region, alongside its significance. In what follows, *significance* is defined as the $-\log_{10}$ *p*-value. The significance of each region was based on the smallest *p*-value in that region.

Region	Significance	Product
rpoB	**5.63**	DNA-directed RNA polymerase subunit beta
PE_PGRS2	1.93	PE-PGRS family protein PE_PGRS2
lpqW	1.78	monoacyl phosphatidylinositol tetramannoside-binding protein LpqW
...	...	...

Manhattan plot

The Figure displays the significance of each kmer against the position in the reference genome to which it mapped. Kmers that did not map are shown at the far right hand side. The Bonferroni-corrected significance threshold is shown as a horizontal black dashed line. The names of significant regions are plotted above. Points are colour-coded in an adjustable manner to display minor allele frequency (MAF), β (direction of effect) or uniqueness of mapping. The MAF threshold can also be removed (although the significance threshold is not updated since we do not recommend reporting low-MAF kmers as significant).

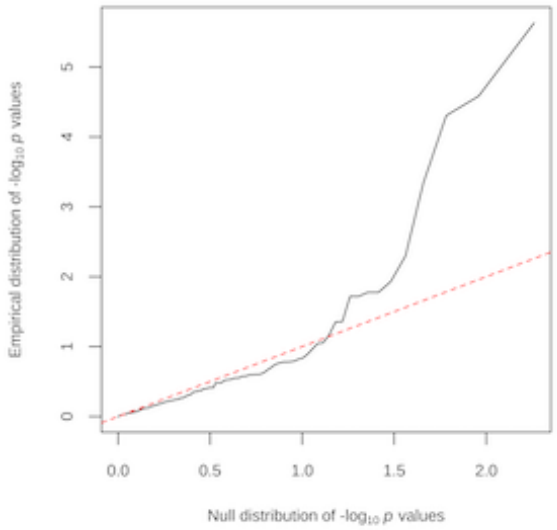


Kmers colour-coded by minor allele frequency.

QQ plots

The QQ plots in the Figure below allow an assessment of whether there were any problems with inflation of significance in the analysis. Inflation is detected by an elevation of the black solid line above the red dashed line at relatively small -$\log_{10}$ *p*-values. An elevation of the black solid line above the red dashed line only at relatively large values (e.g. above the significance threshold) is evidence of association, rather than inflation.

If the black solid line falls below the red dashed line, that may provide evidence of deflation, which occurs when the analysis is under-powered. The removal of low MAF variants is one measure aimed at avoiding deflation by avoiding under-powered tests. Note that the QQ plot is noisier at larger -$\log_{10}$ *p*-values.



QQ plot with maf filter of 0.01.

The report is interactive and contains links to reports on specific regions. For example, the report on *rpoB* is reproduced below:

Kmer GWAS report: *rpoB*

```
Prefix: tb20; KmerType: nucleotide; K: 31; ReferenceGenome: NC_000962.3; MAF: 0.01;
MinCount: 1; AlignIdent: 90; ReportTimeStamp: Wed Oct 26 10:03:59 2022.
```

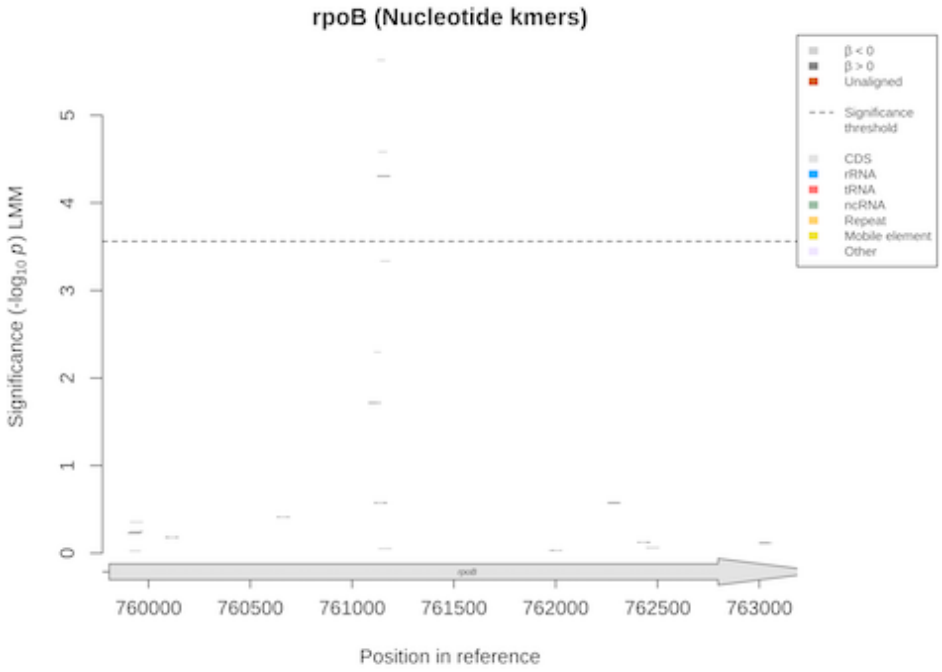
rpoB was the 1st most significant region, with a minimum *p*-value of $10^{-5.63}$.

The user-provided Genbank file lists *rpoB* (Rv0667) as 1172 nucleotides long. It encodes the DNA-directed RNA polymerase subunit beta (protein ID NP_215181.1).

Manhattan plot for *rpoB*

The Figure displays the significance of each kmer against the position in the reference genome to which it mapped, with a focus on *rpoB* . The Bonferroni-corrected significance threshold is shown as a horizontal black dashed line. Annotated features are plotted below. Points are shaded light

($\beta < 0$) or dark ($\beta > 0$) to indicate direction of association, and colour-coded grey (unique) or orange (non-unique) to indicate the quality of mapping. When $\beta > 0$, the presence of the kmer is associated with larger values of the phenotype. The figure can be displayed with or without filtering of kmers below the MAF threshold (although the significance threshold is not updated since we do not recommend reporting low-MAF kmers as significant).

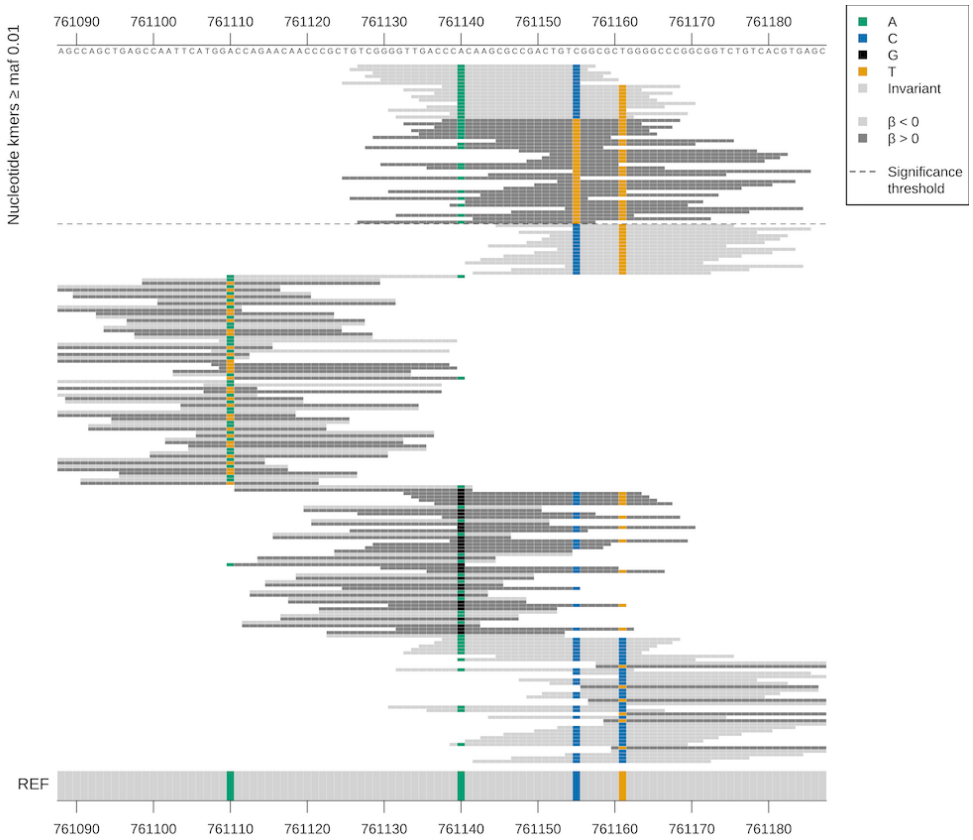


Kmers mapping to the region, filtered by MAF.

High-resolution Earle plots for *rpoB*

The series of Figures below are used to identify the underlying variants tagged by significant kmers. Resembling Manhattan plots, these are high-resolution figures plotting individual kmers against the position to which they mapped in the reference genome, in the region of *rpoB* . The kmers are sorted vertically in order of significance, with the most significant kmers at the top. The horizontal black dashed line demarcates kmers above and below the Bonferroni-corrected significance threshold.

The kmers are shaded light ($\beta < 0$) or dark ($\beta > 0$) to indicate direction of association. Where there is sequence variation relative to the reference genome, individual sites are colour-coded by allele according to the key. The reference allele is indicated at the bottom. Only invariant sites are coloured grey. Use the arrows to scroll through and jump between windows of significance within the region. By default, low-MAF kmers are filtered out. Use the checkbox to remove this filter, which can sometimes assist in interpretation of the signal of association. For instance, in the case of antimicrobial resistance, there are often multiple very low-MAF mutants associated with increased resistance (darker kmers) which can fall below the MAF threshold. These mutants might have evolved independently, and show lower significance than wild types associated with reduced resistance (lighter kmers) because their low frequency reduces statistical power.



kmer	Signif	beta	MAC	qstart	qend	sstart	send	pident	length	mism	gapo	eval
CCGACAGTCGGCGCTTGTGGGTCAACCCCGA	5.63	-6.00	13	1	31	761157	761127	100	31	0	0	12
CGACAGTCGGCGCTTGTGGGTCAACCCCGAC	5.63	-6.00	13	1	31	761156	761126	100	31	0	0	12
CGCCGACAGTCGGCGCTTGTGGGTCAACCCC	5.63	-6.00	13	1	31	761159	761129	100	31	0	0	12
CGGGGTTGACCCACAAGCGCCGACTGTCGGC	5.63	-6.00	13	1	31	761128	761158	100	31	0	0	12
GCGCCGACAGTCGGCGCTTGTGGGTCAACCC	5.63	-6.00	13	1	31	761160	761130	100	31	0	0	12
TGTCGGGGTTGACCCACAAGCGCCGACTGTC	5.63	-6.00	13	1	31	761125	761155	100	31	0	0	12
CCACAAGCGCCGACTGTCGGCGCTGGGGCCC	4.58	-5.64	14	1	31	761138	761168	100	31	0	0	12
…	…	…	…	…	…	…	…	…	…	…	…	…

The Table above provides detailed information on the kmers plotted in the Figure, ordered from most significant (top) to least significant (bottom). In the table, `beta` provides the direction and magnitude of the association between the phenotype and the presence of the kmer, and `MAC` provides the minor allele count (no filter was applied to the Table). The remaining columns were produced by BLAST: `qstart`, `qend`, `sstart` and `send` provide the start and end coordinates of the BLAST match for the query (kmer) and subject (reference genome). The match is further summarized by the `pident` (percent identity), `length`, number of `mism[atches]`, `gapo[pen]` events, and the $\log_{10}$ of the `eval[ue]`.

Having found significant regions, the challenge is then to interpret the signal to understand the possible functional role of genetic variation that is tagged by the significantly associated kmers. One starting point is to blast significant kmers. The report will mention if there are significant kmers that did not map to the user-provided reference genome. A blast analysis is particularly useful to understand these unmapped signals.

In the example dataset, the most significant kmers tag a C→T substitution encoding a non-synonymous S450L change in the rifampicin resistance determining region. This result would be more immediately apparent from the protein-based analysis, which can be run by altering `nextflow.config` so `kmer_type = "protein"` and e.g. `kmer_length = 11`.

OUTPUT

Executes the kmer pipeline end-to-end.
One file ending
.report.html
One directory ending
_kmergenealign_figures
Contains a summary of the GWAS results readable in a web browser, including Manhattan plots, QQ plots, tables of significant regions and kmers.

Before proceeding, save a Nextflow configuration file named *nextflow.config* in the current working directory to specify the analysis parameters. See section nextflow.config for details.

General usage:

```
nextflow kmer_pipeline.nf [--parameter_to_override value] [-resume]
```

Consult the Nextflow documentation for more information.

Tips:

- Launch nextflow from within the `base_dir` sub-directory tree so the nextflow `work` folder and logs are stored alongside the analysis output.
- Keep `nextflow.config` in the `base_dir` for future reference. Avoid overriding parameters on the command line for the same reason.
- **Run a clean analysis**: remove everything in `analysis_dir` and the Nextflow `work` directory before launching.
- Test your setup works beforehand by analysing the example data.

Disclaimer: kmer_pipeline is a Nextflow port of scripts written originally in bash and R for a Univa Grid Engine environment. It does not fully conform to Nextflow design philosophies, particularly in writing to a common directory and omitting input/output files as process arguments. This could cause unexpected behaviour, for example using the Nextflow `-resume` option.

1. Nextflow-inside-Container

`maxp`	Maximum number of cores available for your use on the bare metal machine.
`container_type`	`"none"`
`container_file`	`""`

This is the simplest set-up: launch the container on a 'bare metal' machine using Docker or Singularity, and run Nextflow *inside* the container. Only the container software (Docker or Singularity) needs to be pre-installed.

Run the following command first to launch the Docker container:

```
docker run -it --rm -v BASE_DIR:/home/jovyan CONTAINER_NAME bash
```

Run the following command first to launch the Singularity container:

```
singularity exec --containall --cleanenv --home BASE_DIR:/home/jovyan
CONTAINER_FILE bash
```

Replace BASE_DIR, and CONTAINER_NAME or CONTAINER_FILE as appropriate above.

NB: All user filesystem paths must be given *inside* the container (i.e. via /home/jovyan) when running Nextflow inside the Docker/Singularity container. This affects `nextflow.config` and the `id_file`.

2. Nextflow running Container on a cluster (*recommended*)

`maxp`	Maximum number of cores available for your use on the cluster.
`container_type`	`"singularity" or "docker"`
`container_file`	`"/full/path/to/container_file"` [**for Singularity**] `"container_name:tag"` [**for Docker**]

This is more scalable, but requires pre-installation of both Nextflow and the container software (Docker or Singularity) on the cluster. For a slurm cluster add the following to the bottom of nextflow.config, substituting "short" for the name of a queue (partition) you have access to. For Sun Grid Engine-like systems, replace "slurm" with "sge".

```
process {
      executor = "slurm"
      queue = "short"
}
```

In this setup, Nextflow is to be run directly, without first launching Docker or Singularity.

3. Nextflow on a cluster with no Container (*not recommended, not supported*)

`maxp`	Maximum number of cores available for your use on the cluster.
`container_type`	`"none"`
`container_file`	`""`

This is the most optimized setup, and does not require Docker or Singularity, but instead requires manual installation ***all*** pre-requisite software in the kmer_pipeline, including Nextflow. Refer your system administrators to the Dockerfile to determine your local installation requirements. Warning: this is technical and likely to be time-consuming, and therefore not recommended.

Again the following section is needed at the end of nextflow.config:

```
process {
      executor = "slurm"
      queue = "short"
}
```

substituting "slurm" and "short" as required.

The `nextflow.config` file should be copied into the Nextflow working directory, where it will be detected automatically. Consult the documentation to understand where Nextflow looks for `nextflow.config`.

The `nextflow.config` file is structured into named code blocks, e.g. `params { ... }`. The label is known as the scope. Arguments are specified by assignment e.g. `kmer_length = 11` inside the appropriate scope, e.g. inside `params { ... }`. They can also be defined outside a block by explicitly specifing the scope e.g. `params.kmer_length = 11`. Parameters can be overridden at the command line using double hyphen, e.g. `nextflow kmer_pipeline.nf --kmer_length 11`.

Strings must be quoted. Double quotes allow cross-referencing of parameters using special notation e.g. `analysis_dir = "$base_dir/$output_prefix/kmergwas"`. Single quotes do not allow this cross-referencing.

Besides the `params` scope which specifies pipeline-specific parameters, some Nextflow parameters are specified within the `executor` scope. In what follows, default values are enclosed [as such]. Parameters with default values do not need to be specified in `nextflow.config`.

`params {...} [basic usage]`

Output files

`base_dir`	Directory on the user file system that is parent to all other directories and files involved in the analysis.
`output_prefix`	Filename prefix for output files.
`analysis_dir`	Directory to store output files. Can be specified relative to base_dir, e.g. `"$base_dir/$output_prefix"`

Analysis options

`kmer_type`	`"nucleotide"` or `"protein"`
`kmer_length`	e.g. 31 (*currently the maximum*) or 11 NB: very short lengths can cause mapping to fail

Input files

`id_file`	Tab-delimited text file on the user file system containing a column of sample names with header ‘id’, a column containing paths to the genome assemblies with header ‘paths’ and a column containing the phenotypes with header ‘pheno’.

Species-specific reference genome FASTA and genbank files

`ref_fa`	File path on the user file system for the reference fasta file.
`ref_gb`	File path on the user file system for the reference genbank file.

Deployment

`maxp`	Maximum parallelization. The value should reflect the constraints imposed by the compute environment.
`container_type`	[`"none"`] The type of container in which to run kmer_pipeline: either `"none"`, `"singularity"` or `"docker"`.
`container_file`	[`""`] Must be specified if `container_type != "none"`, to be used in generating `container_cmd`. **Singularity**: the path and filename of the container on the user file system. **Docker**: the name (and tag) of the container to use.

executor {...}

Output files

queueSize	Enter params.maxp to ensure the expected parallelization in most executors.
cpus	Enter params.maxp to ensure the expected parallelization in certain executors.

params {...} [advanced usage]

Analysis options

ntopgenes	[20] Number of the most significant genes or intergenic regions on which to create reports.
minor_allele_threshold	[0.01] Minor allele threshold for excluding extreme-frequency kmers. If the threshold is between 0-0.5, assumed to be a minor allele frequency (MAF) threshold. If the threshold is greater than or equal to 1, assumed to be a minor allele count (MAC) threshold
min_count	[1] Minimum number of genomes a kmer/gene combination must be seen in to be plotted in the Manhattan plot. For genome assemblies, set to 1.
nucmerident	[90] Minimum percentage identity threshold for a nucmer contig alignment to be used to position a kmer, between 0-100.
bowtie_parameters	["--very-sensitive"] Parameters for running bowtie2. If the provided option is not the default, assumes a text file where the lines read in are the bowtie parameters used.
samtools_filter	[10] Bowtie2 mapping quality filter. Samtools is used to remove kmers mapped below this threshold.
blastident	[70] Minimum percentage identity threshold for a BLAST kmer alignment to be kept, between 0-100.

Input files

covariate_file	[""] Gemma formatted covariate file. First column must be a column of 1s for the intercept.

Deployment

container_args	[""] Convenient way to append *additional* arguments to container_cmd, e.g. to specify where to mount temporary directories. For advanced use, edit container_cmd directly.
container_cmd	[*Advanced use only*] Specified automatically from other parameters, but can be overridden by advanced users. Command to execute the container.
container_mount	[*Advanced use only*] Specified automatically from other parameters, but can be overridden by advanced users. Location in the container file system to mount base_dir.
software_file	[*Advanced use only*] Specified automatically, but can be overridden by advanced users for development. File in the user file system containing paths to the pipeline scripts and required software. Needed for container-less installation.

Workflow parameters

`skip1` [`false`] Skip the specified step of the pipeline if `true`.
`skip2`
`skip3`
`skip4`
`skip5`
`skip6`
`skip7`

`kmer_pipeline.nf` also outputs to screen a list of implied parameters, constructed automatically from the parameters detailed above. While some of these could be overridden for debugging purposes, that is not recommended.

Step 1 Count kmers

Information on individual steps of the pipeline is for reference only, knowledge of their usage is not necessary to run the pipeline.

Input for kmer counting is assembly contigs, not sequencing reads. Kmers are counted as present if seen once in a genome.

OUTPUT

Kmers for each combination of kmer type (nucleotide/protein) and kmer length per genome.

If a protein kmer analysis is performed, an additional file will be produced for each genome containing the contigs translated into all six possible reading frames in a 'translated_contigs' subdirectory within 'analysis_dir'. A file will be produced containing all protein kmers in a separate subdirectory for each kmer length ('e.g. protein11') in 'analysis_dir'.

For nucleotide kmers, the kmers are counted directly from the genome assemblies and stored in a separate subdirectory within 'analysis_dir' for each kmer length (e.g. 'nucleotide 31').

Usage:

```
countkmers.Rscript task_id id_file analysis_dir output_prefix
software_file [analyses_list]
```

Arguments

`task_id`	The task number to run (1…n).
`id_file`	A text file containing a column of sample names with header 'id', a column containing paths to the genome assemblies with header 'paths' and a column containing the phenotypes with header 'pheno'. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
`analysis_dir`	Directory location for the analysis.
`output_prefix`	Output file prefix.
`software_file`	File containing paths to the pipeline scripts and required software described on page 44.
`analyses_list`	*Optional argument. Default = counts 31bp nucleotide kmers.* A text file specifying the analyses to run. The file should contain a column containing the types of kmer to be analysed 'nucleotide' or 'protein' with column header 'kmertype' and a column with the kmer length to correspond with each variant type to be tested with column header 'kmerlength'. If not specified, the pipeline will default to 31bp length nucleotide kmers. Example file: script_dir/example/analysistype_nucleotide31_protein11.txt

Step 2 Create unique kmer list

Merges all kmers created in step 1 found in the subdirectory ‘analysis_dir/kmertypekmerlength/’ for the samples in id_file.

OUTPUT

One file ending:
‘**.kmermerge.txt.gz**’
All kmers present at least once across the samples in ‘id_file’ within the directory ‘analysis_dir’. Must be run for each kmer type and length separately.

Merges in stages, first merges into p files, then performs subsequent merges until one output file is produced.

Temporary files are created in the run directory and deleted.

Usage:

```
createfullkmerlist.Rscript task_id n p output_prefix analysis_dir
id_file kmer_type kmer_length software_file
```

Arguments

`task_id`	The task number to run (1…p).
`n`	Total number of samples.
`p`	Total number of processes to run at one time.
`output_prefix`	Output file prefix.
`analysis_dir`	Directory location for the analysis. Where the subdirectories will be created to store the kmer files.
`id_file`	A text file containing a column of sample names with header ‘id’, a column containing paths to the genome assemblies with header ‘paths’ and a column containing the phenotypes with header ‘pheno’. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
`kmer_type`	Either ‘protein’ or ‘nucleotide’.
`kmer_length`	Kmer length.
`software_file`	File containing paths to the pipeline scripts and required software described on page 44.

OUTPUT

Patterns

Patterns are first created for batches of kmers and stored in a subdirectory 'kmertypekmerlength_patternbatches' for use in a later step. The batches are merged into one set of patterns in stages creating the files ending:

.patternmerge.patternKey.txt.gz

Unique presence/absence patterns. Each line is a separate pattern with 0 representing absence and 1 presence of a kmer. The 0/1 order is determined by the order of the samples in 'id_file'.

.patternmerge.patternIndex.txt.gz

A 0-based index the length of the total number of kmers. Each line describes the presence/absence pattern in .patternmerge.patternKey.txt.gz for the corresponding kmer in the file ending .kmermerge.txt.gz.

.patternmerge.presenceCount.txt.gz (phenotype dependent file)

For each pattern, the sum of the number of genomes kmers with that pattern are present in. If there are any NAs within the phenotype file, those samples are not included when determining the counts.

Kinship matrix

Kinship matrices are first created for each batch of patterns, then merged into one kinship matrix in stages creating the files ending:

.kinship.txt.gz

Kinship matrix file. Rows and columns are ordered according to the order of samples in 'id_file'.

.kinshipWeight.txt

Contains the number of kmers used to create the full kinship file, this should be equal to the total number of kmers in .kmermerge.txt.gz.

Temporary files are created in the run directory and deleted.

The standard output will be written to a file for each process, and the standard error if any errors occur. Recommend running in a separate run directory due to the large number of stdout and stderr files.

The presence/absence patterns and kinship matrix files are created for the full set of samples included in id_file, ignoring the phenotype column. If there are any NAs in the phenotype column, these samples are still included in the patterns, and kinship matrix. The pattern counts are determined for just the samples with a non NA phenotype. To get the pattern counts for all samples or those with non NA phenotypes, pattern2presencecount.Rscript can be run separately adjusting the input includeNA.

If the patterns file and kinship matrix have been successfully created, the pattern batches directory can be deleted.

Usage:

```
stringlist2patternandkinship.Rscript task_id p id_file fullkmerlistfile
kmercountslistfile analysis_dir output_prefix kmertype software_file
[kmer_length=31 mincount=5]
```

Arguments

task_id	The task number to run (1…p).
p	Number of batches to split the kmer patterns into. Also the maximum number of processes to run at the same time.
id_file	A text file containing a column of sample names with header ‘id’, a column containing paths to the genome assemblies with header ‘paths’ and a column containing the phenotypes with header ‘pheno’. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
fullkmerlistfile	File containing the full list of unique kmers in the dataset, created in step 3. File ending .kmermerge.txt.gz.
kmercountslistfile	File containing the paths to all kmer count files created in step 2, ending kmers_filepaths.txt.
analysis_dir	Directory location for the analysis. Location for the final output files and where the subdirectory will be created to store the pattern batches.
output_prefix	Output file prefix.
kmertype	Either ‘protein’ or ‘nucleotide’.
software_file	File containing paths to the pipeline scripts and required software described on page 44.
kmer_length	Kmer length.
mincount	Minimum number of times a kmer has to be present in a sample to be counted as present. Set this to 1 as the kmers have been counted from assemblies.

Step 4 Run GEMMA

OUTPUT

The kmer patterns are split into 'p' batches and GEMMA is run separately for each batch of patterns. The gemma output files will be stored within a subdirectory kmer_typekmer_length_gemma/output within 'analysis_dir'. Output files for each batch ending:

.assoc.txt.gz
: GEMMA output file containing the pattern number, p-values and log likelihood under the alternative.

.pval.txt.gz
: The likelihood ratio test p-value column extracted from the .assoc.txt.gz file.

.log.txt.gz
: GEMMA log file containing the heritability estimate and standard error.

Temporary files are created in the run directory and deleted.

The standard output will be written to a file for each process, and the standard error if any errors occur. Recommend running in a separate run directory.

Usage:

```
rungemma.Rscript task_id p kmerFilePrefix id_file output_prefix
analysis_dir kmertype kmer_length software_file [covariate_file]
```

Arguments

`task_id`	The task number to run (1…p).
`p`	Number of batches to split the GEMMA runs into. Also the number of processes run at the same time.
`id_file`	A text file containing a column of sample names with header 'id', a column containing paths to the genome assemblies with header 'paths' and a column containing the phenotypes with header 'pheno'. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
`output_prefix`	Output file prefix.
`analysis_dir`	Directory location for the analysis. Where the subdirectory will be created to store the gemma output.
`kmertype`	Either 'protein' or 'nucleotide'.
`kmer_length`	Kmer length.
`software_file`	File containing paths to the pipeline scripts and required software described on page 44.
`covariate_file`	*Optional input.* Gemma formatted covariate file. First column must be a column of 1s for the intercept.

Step 5 Run contig alignment

For each assembly, contigs are aligned to the specified reference genome using nucmer.

OUTPUT

The reference genome is read in and just the CDS are kept. An ID is assigned to each of the genes, 1-n. Intergenic regions are then assigned an ID. If a gene does not overlap with the preceeding gene, then the intergenic region is assigned an ID, these begin at n+1. This produces the output file ending:

_gene_id_name_lookup.txt

Contains the ID number used for each gene and intergenic region. Intergenic regions are written by joining the two flanking genes with ':'.

One file per genome is produced in the subdirectory 'kmer_typekmer_length_kmergenealign' with the kmer/gene combination files ending:

.kmer_list_gene_IDs.txt.gz

First column is the 1-based unique kmer/gene combinations. The kmers being those in .kmermerge.txt.gz. E.g. for the first kmer in .kmermerge.txt.gz aligned to gene 5 it would be '1,5'.

Second column is a dummy count to be in the correct format for the next stage, and can be ignored.

One file is produced in the same subdirectory ending:

kmergenecombination_filepaths.txt

Containing paths to all output files. This file is used for the next step.

Usage:

```
kmercontigalignonly.Rscript task_id n output_prefix output_dir id_file
ref_fa ref_gb kmer_type kmer_length nucmerident kmerSeqFile
software_file [kstart=9 kend=100]
```

Additionally to merge the kmer/gene alignments (can be run separately – see next step):

```
kmercontigalign.Rscript task_id n output_prefix output_dir id_file
ref_fa ref_gb kmer_type kmer_length nucmerident kmerSeqFile
software_file [kstart=9 kend=100]
```

Arguments

`task_id`	The task number to run (1…n).
`n`	Number of samples.
`output_prefix`	Output file prefix.
`analysis_dir`	Directory location for the analysis. Where the subdirectory 'kmer_typekmer_length_kmergenealign' will be created to store the output files.
`id_file`	A text file containing a column of sample names with header 'id', a column containing paths to the genome assemblies with header 'paths' and a column containing the phenotypes with header 'pheno'. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
`ref_fa`	File path to the reference fasta file.
`ref_gb`	File path to the reference genbank file.
`kmer_type`	Either 'protein' or 'nucleotide'.

`kmer_length`	Kmer length. If set to 0 then the kmer length is assumed to be variable. If kstart and kend are not set, assuming variable kmer lengths between 9-100 bases long.
`nucmerident`	Minimum percentage identity threshold for a nucmer contig alignment to be used to position a kmer, between 0-100.
`kmerSeqFile`	Output file from step 2 ending '.kmermerge.txt.gz'
`software_file`	File containing paths to the pipeline scripts and required software described on page 44.
`kstart`	*Optional input.* If kmer_length is set to 0 (meaning variable kmer lengths) the minimum kmer length to use. Default = 9.
`kend`	*Optional input.* If kmer_length is set to 0 (meaning variable kmer lengths) the maximum kmer length to use. Default = 100.

Step 5A Merge kmer/gene alignments

This script can be run as part of Step 5 but can be run separately.

OUTPUT

Two files ending:

.kmeralignmerge.txt.gz

Containing the merged kmer/gene combinations present at least once across all files in 'input_files'. Numbers are as in step 5.

.kmeralignmerge.count.txt.gz

Containing the number of genomes each kmer/gene combination is found in to allow for filtering in later steps.

Usage:

```
kmercontigalignmerge.Rscript task_id n p output_prefix analysis_dir
input_files kmer_type kmer_length ref_fa nucmerident
```

Arguments

`task_id`	The task number to run (1…p).
`n`	Number of samples.
`p`	Maximum number of processes to run at the same time, should be smaller than n/2.
`stdout_dir`	Directory to write stdout. If -o is not supplied, stdout will be written to the present working directory.
`stderr_dir`	Directory to write stderr. If -e is not supplied, stderr will be written to the present working directory.
`output_prefix`	Output file prefix.
`analysis_dir`	Directory location for the analysis.
`input_files`	A file containing the paths to the kmer/gene alignment combinations per sample. Created in step 5 ending 'kmergenecombination_filepaths.txt'.
`kmer_type`	Either 'protein' or 'nucleotide'.
`kmer_length`	Kmer length. If set to 0 then the kmer length is assumed to be variable. If kstart and kend are not set, assuming variable kmer lengths between 9-100 bases long.
`ref_fa`	File path to the reference fasta file.
`nucmerident`	Minimum percentage identity threshold for a nucmer contig alignment to be used to position a kmer used in step 5, between 0-100.

Step 6 Plot figures using contig alignment positions

Assumptions:
Reads in the kmer files with the provided prefix ('kmerfilePrefix'). This includes the file ending '.patternmerge.presenceCount.txt.gz' which is a phenotype dependent file when some phenotypes are set to NA, see step 3.

OUTPUT

Files produced in the subdirectory 'kmer_typekmer_length_kmergenealign_figures'.
Files ending:

QQplot_allkmers.png
QQ plot for all kmers

QQplot_ma*.png
QQ plot for all kmers above the MAC/MAF threshold

Manhattan_alignCOL_ma*.png
Manhattan plot coloured by whether the kmer/gene assignment was unique (grey) or if the kmer aligned to multiple genes or intergenic regions (red). Only plots kmers above the MAC/MAF threshold.

Manhattan_betaCOL_ma*.png
Manhattan plot coloured by the beta estimate for the significant kmers. Strength of the colour indicates the magnitude of the beta estimate. Only plots kmers above the MAC/MAF threshold.

Manhattan_mafCOL_ma*.png
Manhattan plot coloured by the MAF category. Only plots kmers above the MAC/MAF threshold.

Manhattan_mafCOL_allkmers.png
Manhattan plot coloured by the MAF category. All kmers are plotted.

In the subdirectory 'kmerfiles':

unaligned_kmersandpvals.txt
All kmers with no gene or intergenic region assigned to them. Columns include kmer, $-\log_{10} p$, beta estimate and MAC.

The x-axis positions for the genome-wide Manhattan plots are from the kmer/gene combinations from step 5 above the count threshold. The kmers are plotted at the midpoint of the gene or intergenic region they were assigned to, which could be multiple. Manhattan plots with file names ending "_ylim50.png" cut the y-axis at 50 where the significance of the top kmers is above 100.

The following files are produced for either:

- The top 20 most significant genes or intergenic regions, or
- The genes provided in annotateGeneFile

In the subdirectory 'kmerfiles'

kmersandpvals.txt
Contains all kmers assigned to each gene or intergenic region by nucmer, plus the columns $-\log_{10} p$, beta estimate and MAC.

OUTPUT

In the subdirectory ‘alignments’ followed by gene or intergenic region name:

blast_results.txt

Results from running blast on the kmers in the above kmersandpvals.txt file. For protein kmers there will be a separate file for each of the six possible reading frames.

no_blast_result_or_poor_alignment.txt

Subset of the above file kmersandpvals.txt. The kmers that either did not align to any reading frame or aligned poorly using blast. If all kmers aligned well this file is not produced.

Manhattan_allkmers.png

A close up Manhattan plot of a particular gene/IR. The kmers that were assigned to the gene/IR by nucmer are realigned to the gene/IR using blast. All kmers are plotted.
For protein kmers, this will be plotted for the correct reading frame (for intergenic regions this is taken to be frame one on the forward strand) and for all six possible frames combined.

Manhattan_ma*.png

As above but only kmers above the MAC/MAF threshold.

alignment.png

A close up of the kmers aligned to a particular gene/IR compared to the reference. Kmers are coloured by their direction of effect and by variants present. Figures are created in a sliding window across significant regions of the gene/IR, defined as regions with significant kmers above the MAC/MAF threshold (if override_signif=FALSE) or across the whole gene (if override_signif=TRUE). For the protein kmers, this is just plotted for the correct reading frame (for intergenic regions this is taken to be frame one on the forward strand).

ma*_alignment.png

As above but only kmers above the MAC/MAF threshold.

In the subdirectory ‘alignments’:

all_top_genes_significant_kmers_per_alignment_plot.txt

Contains the kmer sequence, $-\log_{10} p$, beta, MAC, MAF, and leftmost position for the kmers shown in the alignment.png figures. All kmers are included, not just those above the MAC/MAF threshold. This will contain all significant kmers per alignment figure (if override_signif=FALSE) or all kmers (if override_signif=TRUE).

Usage:

```
plotManhattan.Rscript output_prefix analysis_dir kmerfilePrefix ref_gb
ref_fa gene_lookup_file id_file nucmerident min_count kmer_type
kmer_length minor_allele_threshold software_file blastident ngenes
[annotateGeneFile=NULL override_signif=FALSE]
```

Arguments

`output_prefix`	Output file prefix.
`analysis_dir`	Directory location for the analysis. Where the subdirectory ‘kmer_typekmer_length_figures’ will be created to store the output files.
`kmerfilePrefix`	Prefix, including path, to the kmer files created in step 3. E.g. /path/to/prefix (where the full files are e.g. /path/to/prefix.patternmerge.patternKey.txt.gz, /path/to/prefix.patternmerge.patternIndex.txt.gz)
`ref_gb`	File path to the reference genbank file.
`ref_fa`	File path to the reference fasta file.
`gene_lookup_file`	File created in step 5 ending ‘gene_id_name_lookup.txt’ in the subdirectory ending ‘_kmergenealign’
`id_file`	A text file containing a column of sample names with header ‘id’, a column containing paths to the genome assemblies with header ‘paths’

	and a column containing the phenotypes with header ‘pheno’. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
nucmerident	Minimum percentage identity threshold for a contig alignment to be used to position a kmer used in step 5, between 0-100.
min_count	Minimum number of genomes a kmer/gene combination must be seen in to be plotted in the Manhattan plot.
kmer_type	Either ‘protein’ or ‘nucleotide’.
kmer_length	Kmer length.
minor_allele_threshold	*Optional argument.* Minor allele threshold to exclude kmers below the threshold. If the threshold is between 0-0.5, assumed to be a minor allele frequency (MAF) threshold. If the threshold is greater than or equal to 1, assumed to be a minor allele count (MAC) threshold.
software_file	File containing paths to the pipeline scripts and required software described on page 44.
blastident	Minimum percentage identity threshold for a BLAST kmer alignment to be kept, between 0-100.
ngenes	Specifies the number of top hit genes on which to report.
annotateGeneFile	*Optional argument.* File containing a list of genes or intergenic regions to annotate. Intergenic regions should be written as the two flanking genes separated by a colon, e.g. “geneA:geneB”. Example file: script_dir/example/example_annotate_gene_file.txt
override_signif	*Optional argument.* If a file is provided for annotateGeneFile, should all kmer alignments be plotted for the genes even if they are not significant. Default = FALSE.

Step 5B Run bowtie2 (nucleotide kmers only)

OUTPUT

In the subdirectory 'kmer_typekmer_length_bowtie2mapping'.
Files ending:
.bt2
Bowtie2 formatted reference files.
referencename.gz
Mapping results for all kmers.
In the directory analysis_dir:
File ending:
bowtie2map.txt.gz
Mapping results for all kmers that passed the quality filter.

Usage:

```
runbowtie.Rscript output_prefix analysis_dir kmerfilePrefix ref_fa
kmer_type kmer_length software_file [bowtie_parameters=--very-sensitive
samtools_filter=10]
```

Arguments

`output_prefix`	Output file prefix.
`analysis_dir`	Directory location for the analysis. Where the subdirectory 'kmer_typekmer_length_figures' will be created to store the output files.
`kmerfilePrefix`	Prefix, including path, to the kmer files created in step 3. E.g. /path/to/prefix (where the full files are e.g. /path/to/prefix.patternmerge.patternKey.txt.gz, /path/to/prefix.patternmerge.patternIndex.txt.gz)
`ref_fa`	File path to the reference fasta file.
`kmer_type`	Either 'protein' or 'nucleotide'.
`kmer_length`	Kmer length.
`software_file`	File containing paths to the pipeline scripts and required software described on page 44.
`bowtie_parameters`	*Optional argument.* Parameters for running bowtie2. Default = "—very-sensitive". If the provided option is not the default, assumes a text file where the line read in are the bowtie parameters used.
`samtools_filter`	*Optional argument.* Bowtie2 mapping quality filter. Samtools is used to remove kmers mapped below this threshold. Default = 10.

Step 6B Plot figures using bowtie2 mapping positions (nucleotide kmers only)

Assumptions:
Reads in the kmer files with the provided prefix ('kmerfilePrefix'). This includes the file ending '.patternmerge.presenceCount.txt.gz' which is a phenotype dependent file when some phenotypes are set to NA, see step 3.

OUTPUT

Files produced in the subdirectory 'kmer_typekmer_length_bowtie2mapping_figures'.
Files ending:

QQplot_allkmers.png
QQ plot for all kmers

QQplot_ma*.png
QQ plot for all kmers above the MAC/MAF threshold

Manhattan_alignCOL_ma*.png
Manhattan plot coloured by whether the kmer/gene assignment was unique (grey) or if the kmer aligned to multiple genes or intergenic regions (red). Only plots kmers above the MAC/MAF threshold.

Manhattan_betaCOL_ma*.png
Manhattan plot coloured by the beta estimate for the significant kmers. Strength of the colour indicates the magnitude of the beta estimate. Only plots kmers above the MAC/MAF threshold.

Manhattan_mafCOL_ma*.png
Manhattan plot coloured by the MAF category. Only plots kmers above the MAC/MAF threshold.

Manhattan_mafCOL_allkmers.png
Manhattan plot coloured by the MAF category. All kmers are plotted.

In the subdirectory 'kmerfiles':

unaligned_kmersandpvals.txt
All kmers with no gene or intergenic region assigned to it. Columns include kmer, $-\log_{10} p$, beta estimate and MAC.

The x-axis positions for the genome-wide Manhattan plots are from running bowtie2 in step 5B above the count threshold. Depending on the bowtie2 settings used, a kmer could be plotted more than once. If the default bowtie2 setting was used then a kmer will just be plotted once in the best mapping position. Manhattan plots with file names ending "_ylim50.png" cut the y-axis at 50 where the significance of the top kmers is above 100.

The following files are produced for either:

- The top 20 most significant genes or intergenic regions, or
- The genes provided in annotateGeneFile

In the subdirectory 'kmerfiles':

kmersandpvals.txt
Contains all kmers that mapped to each gene or intergenic region using bowtie2, plus the columns $-\log_{10} p$, beta estimate and MAC. The leftmost mapping position is used as the kmer position.

OUTPUT

In the subdirectory ‘alignments’ followed by gene or intergenic region name:

blast_results.txt
Results from running blast on the kmers in the above kmersandpvals.txt file.

no_blast_result_or_poor_alignment.txt
Subset of the above file kmersandpvals.txt. The kmers that either did not align or aligned poorly using blast. If all kmers aligned well this file is not produced.

Manhattan_allkmers.png
A close up Manhattan plot of a particular gene/IR. The kmers that were assigned to the gene/IR by nucmer are realigned to the gene/IR using blast. All kmers are plotted.
For protein kmers, this will be plotted for the correct reading frame (for intergenic regions this is taken to be frame one on the forward strand) and for all six possible frames.

Manhattan_ma*.png
As above but only kmers above the MAC/MAF threshold.**alignment.png**

alignment.png
A close up of the kmers aligned to a particular gene/IR compared to the reference. Kmers are coloured by their direction of effect and by variants present. Figures are made in a sliding window across significant regions of the gene/IR, defined as regions with significant kmers above the MAC/MAF threshold (if override_signif=FALSE) or across the whole gene (if override_signif=TRUE).

ma*_alignment.png
As above but only kmers above the MAC/MAF threshold.

In the subdirectory ‘alignments’:

all_top_genes_significant_kmers_per_alignment_plot.txt
Contains the kmer sequence, $-\log_{10} p$, beta, MAC, MAF, and leftmost position for the kmers shown in the alignment.png figures. All kmers are included, not just those above the MAC/MAF threshold. This will contain all significant kmers per alignment figure (if override_signif=FALSE) or all kmers (if override_signif=TRUE).

Usage:

```
plotManhattanbowtie.Rscript output_prefix analysis_dir kmerfilePrefix
ref_gb ref_fa id_file kmer_type kmer_length minor_allele_threshold
samtools_filter software_file blastident ngenes [annotateGeneFile=NULL
override_signif=FALSE]
```

Arguments

output_prefix	Output file prefix.
analysis_dir	Directory location for the analysis. Where the subdirectory ‘kmer_typekmer_length_figures’ will be created to store the output files.
kmerfilePrefix	Prefix, including path, to the kmer files created in step 3. E.g. /path/to/prefix (where the full files are e.g. /path/to/prefix.patternmerge.patternKey.txt.gz, /path/to/prefix.patternmerge.patternIndex.txt.gz)
ref_gb	File path to the reference genbank file.
ref_fa	File path to the reference fasta file.
id_file	A text file containing a column of sample names with header ‘id’, a column containing paths to the genome assemblies with header ‘paths’ and a column containing the phenotypes with header ‘pheno’. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
kmer_type	Either ‘protein’ or ‘nucleotide’.

`kmer_length`	Kmer length.
`minor_allele_threshold`	*Optional argument.* Minor allele threshold to exclude kmers below the threshold. If the threshold is between 0-0.5, assumed to be a minor allele frequency (MAF) threshold. If the threshold is greater than or equal to 1, assumed to be a minor allele count (MAC) threshold.
`samtools_filter`	Bowtie2 mapping quality filter used in step 5B.
`software_file`	File containing paths to the pipeline scripts and required software described on page 44.
`blastident`	Minimum percentage identity threshold for a BLAST kmer alignment to be kept, between 0-100.
`ngenes`	Specifies the number of top hit genes on which to report.
`annotateGeneFile`	*Optional argument.* File containing a list of genes or intergenic regions to annotate. Intergenic regions should be written as the two flanking genes separated by a colon, e.g. “geneA:geneB”.
`override_signif`	*Optional argument.* If a file is provided for annotateGeneFile, should all kmer alignments be plotted for the genes even if they are not significant. Default = FALSE.

Step 7A Generate a kmer GWAS report

This script is run as part of the main pipeline but can be run separately.

OUTPUT

report.css
report.js
A file ending:
.report.html
Containing kmer GWAS report summary.

Usage:

```
gen-report.Rscript prefix anatype k refname ref_gb maf alignident
mincount ngenes srcdir outdir logdir
```

Arguments

`prefix`	Output file prefix.
`anatype`	Either 'protein' or 'nucleotide'.
`k`	Kmer length. If set to 0 then the kmer length is assumed to be variable. If kstart and kend are not set, assuming variable kmer lengths between 9-100 bases long.
`refname`	Name of the reference genome.
`ref_gb`	File path to the reference genbank file.
`maf`	Minor allele threshold to exclude kmers below the threshold. If the threshold is between 0-0.5, assumed to be a minor allele frequency (MAF) threshold. If the threshold is greater than or equal to 1, assumed to be a minor allele count (MAC) threshold.
`alignident`	Minimum percentage identity threshold for a nucmer contig alignment to be used to position a kmer, between 0-100.
`mincount`	Minimum number of times a kmer has to be present in a sample to be counted as present. Set this to 1 as the kmers have been counted from assemblies.
`ngenes`	Specifies the number of top hit genes on which to report.
`srcdir`	Script directory specified in software file.
`outdir`	Directory location for the analysis. Where the subdirectory 'kmer_typekmer_length_figures' will be created to store the output files.
`logdir`	Log directory to read stdout from GEMMA.

Step 7B Generate a kmer GWAS report for a specific gene

This script is run as part of the main pipeline but can be run separately.

OUTPUT

A file ending:
.report_*genename*.html
Containing kmer GWAS report for a specific gene.

Usage:

```
gen-gene-report.Rscript    hit_num prefix anatype k refname ref_gb maf
alignident mincount srcdir outdir logdir
```

or, for the protein kmer GWAS:

```
gen-protein-report.Rscript hit_num prefix anatype k refname ref_gb maf
alignident mincount srcdir outdir logdir
```

Arguments

`hit_num`	Specifies the rank of the top hit on which to report.
`prefix`	Output file prefix.
`anatype`	Either 'protein' or 'nucleotide'.
`k`	Kmer length. If set to 0 then the kmer length is assumed to be variable. If kstart and kend are not set, assuming variable kmer lengths between 9-100 bases long.
`refname`	Name of the reference genome.
`ref_gb`	File path to the reference genbank file.
`maf`	Minor allele threshold to exclude kmers below the threshold. If the threshold is between 0-0.5, assumed to be a minor allele frequency (MAF) threshold. If the threshold is greater than or equal to 1, assumed to be a minor allele count (MAC) threshold.
`alignident`	Minimum percentage identity threshold for a nucmer contig alignment to be used to position a kmer, between 0-100.
`mincount`	Minimum number of times a kmer has to be present in a sample to be counted as present. Set this to 1 as the kmers have been counted from assemblies.
`srcdir`	Script directory specified in software file.
`outdir`	Directory location for the analysis. Where the subdirectory 'kmer_typekmer_length_figures' will be created to store the output files.
`logdir`	Log directory to read stdout from GEMMA.

Step 7C Generate a kmer GWAS report for unmapped kmers

This script is run as part of the main pipeline but can be run separately.

OUTPUT

A file ending:
.report_unmapped.html
Containing kmer GWAS report for unmapped reads.

Usage:

```
gen-unmapped-report.Rscript prefix anatype k refname ref_gb maf
alignident mincount srcdir outdir logdir
```

Arguments

`prefix`	Output file prefix.
`anatype`	Either 'protein' or 'nucleotide'.
`k`	Kmer length. If set to 0 then the kmer length is assumed to be variable. If kstart and kend are not set, assuming variable kmer lengths between 9-100 bases long.
`refname`	Name of the reference genome.
`ref_gb`	File path to the reference genbank file.
`maf`	Minor allele threshold to exclude kmers below the threshold. If the threshold is between 0-0.5, assumed to be a minor allele frequency (MAF) threshold. If the threshold is greater than or equal to 1, assumed to be a minor allele count (MAC) threshold.
`alignident`	Minimum percentage identity threshold for a nucmer contig alignment to be used to position a kmer, between 0-100.
`mincount`	Minimum number of times a kmer has to be present in a sample to be counted as present. Set this to 1 as the kmers have been counted from assemblies.
`srcdir`	Script directory specified in software file.
`outdir`	Directory location for the analysis. Where the subdirectory 'kmer_typekmer_length_figures' will be created to store the output files.
`logdir`	Log directory to read stdout from GEMMA.

Get kmer presence counts

This script is run as part of the main pipeline but can be run separately.

OUTPUT

.patternmerge.presenceCount.txt.gz
For each pattern, the sum of the number of genomes kmers with that pattern are present in. If includeNA is set to FALSE and some phenotypes are NA in the phenotype file, the presence counts will be calculated excluding those samples.

Usage:

```
pattern2presencecount.Rscript kmerFilePrefix output_dir id_file
[includeNA=TRUE]
```

Arguments

`stdout_dir`	Directory to write stdout. If -o is not supplied, stdout will be written to the present working directory.
`stderr_dir`	Directory to write stderr. If -e is not supplied, stderr will be written to the present working directory.
`kmerFilePrefix`	Prefix, including path, to the kmer files created in step 3. E.g. /path/to/prefix (where the full files are /path/to/prefix.patternmerge.patternKey.txt.gz, /path/to/prefix.patternmerge.patternIndex.txt.gz, /path/to/prefix.patternmerge.patternKeySize.txt)
`output_dir`	Directory location for the final output file.
`id_file`	A text file containing a column of sample names with header ‘id’, a column containing paths to the genome assemblies with header ‘paths’ and a column containing the phenotypes with header ‘pheno’. Example file: script_dir/example/ saur12_example_id_path_pheno.txt.
`includeNA`	Optional argument specifying whether to count presence across all samples (TRUE) or just those with a non NA phenotype (FALSE). Default = TRUE.

Knowledge of the following is not required when running the pipeline using containers.

The table below describes the required contents of the software file. The first eight are required for all analyses. Mummer is required if annotating kmers by aligning the contigs to the reference genome using nucmer. Bowtie2 and samtools are required if annotating kmers by mapping the kmers themselves to the reference genome.

Example file in the scripts example subdirectory: pipeline_software_location.txt

Name in software file	**Version included with Docker image***
scriptpath	github.com/danny-wilson/kmer_pipeline
R	v4.1.3
dsk	v2.3.3
dsk2ascii	v2.3.3
gemma	github.com/danny-wilson/gemma0.93b
gemma_libraries	Library location. /usr/lib
blast	v2.9.0-2
genoPlotR	0.8.11
Required to get initial kmer positions by aligning contigs to the reference genome (steps 5-6):	
mummer	Directory containing mummer executables. 3.23+dfsg-4build1
Required to get initial kmer positions by mapping kmers to the reference genome (steps 5B-6B):	
bowtie2	Directory containing bowtie2 executables. 2.3.5.1-6build1
samtools	1.15.1

* Consult the Dockerfile for changes.

Testing

The Docker examples were tested on Amazon Web Services c5.large EC2 instance with 2 CPUs (Intel® Xeon® @ 3.00GHz), 4 GiB RAM and 10 GiB storage, using Docker version 18.03.0-ce, build 0520e24, git version 2.7.4, Java openjdk version 11.0.14 2022-01-18 and Nextflow version 22.10.0 build 5827 on Ubuntu 16.04.2 LTS (Xenial). Docker on a cluster has not been tested.

The Singularity examples were tested on the University of Oxford Biomedical Research Computing facility, which is supported by the Wellcome Trust Core Award Grant Number 203141/Z/16/Z and the NIHR Oxford BRC, with 30 CPUs (Intel® Xeon® @ 2.40-2.60GHz) and 16 GiB RAM, using Singularity version 3.8.7-1.el7, git version 1.8.3.1, Java openjdk 11.0.2 2019-01-15 and Nextflow version 22.04.0 build 5697 on CentOS Linux release 7.9.2009 (Core).

Full list of pipeline scripts

Full list of R scripts and C++ executables callable by the Nextflow script `kmer_pipeline.nf`

Rscripts
countkmers.Rscript
createfullkmerlist.Rscript
stringlist2patternandkinship.Rscript
rungemma.Rscript
kmercontigalign.Rscript
plotManhattan.Rscript
runbowtie.Rscript
plotManhattanbowtie.Rscript
proteinkmermerge.Rscript
nucleotidekmermerge.Rscript
pattern2presencecount.Rscript
sequence_functions.R
Manhattan_functions.R
alignmentfunctions.R
kmercontigalignmerge.Rscript
gen-report.Rscript
gen-gene-report.Rscript
gen-protein-report.Rscript
gen-unmapped-report.Rscript
get_ref_name.Rscript

Executable C++
sort_strings
stringlist2pattern
kmerlist2pattern
patternmerge
patterncounts
pattern2kinship