gtImputation (Genotype Imputation)

v0.16

Dpto. Sistemas y Recursos Naturales
ETSI Montes, Forestal y del Medio Natural
Universidad Politécnica de Madrid

https://github.com/ggfhf/

Table of contents

Disclaimer

The software package gtImputation (Genotype Imputation) is available for free download from the GitHub software repository (https://github.com/GGFHF/gtImputation) under GNU General Public License v3.0.

Disclaimer

Introduction

Reduced representation sequencing methodologies allow genotyping of many specimens at once and are extensively used in many fields of genomics. The output of sequencing platforms is processed to call variants at specific polymorphic sites of the individual genomes, ant all the information is stored in VCF (Variant Call Format) files. In these files, the format of the records with variant information consists of a series of tab-delimited fields including: the REF field that indicates the base or bases of the reference allele, the ALT field that includes the base or bases of the alternative alleles, the FORMAT field that contains a list of the data reported for each sample and n SAMPLE fields with the values of the data detailed in FORMAT for each sample. The genotype is one of the data reported and is represented by the two encoded alleles separated by a '/' character (non-phased genotypes) or the '|' character (phased genotypes). The allele coding is denoted by a '0' for the reference allele, an '1' for the first alternative allele, a '2' for the second alternative allele, and so on. In the case where an allele in a sample is missing data it is represented by a '.' character.

One of the caveats of current methodologies of reduced representation genome approaches is that they produce large amounts of missing data that may impact statistical inference and introduce bias in the outcome of experiments that use these methods. There are several strategies to reduce the impact of missing data in genomic datasets. One solution is to remove variants containing missing data, but, depending on the outcome of the experiment, this can lead to loss of accuracy in genomic inference. Another approach is to deduce (impute) the genotype that each missing data should have using validated references through frequentist or machine learning approaches.

gtImputation

gtImputation is an application designed to determine the genotypes of missing data in a VCF or tabular format file using a machine learning procedure based on self-organizing maps (SOMs) called **SOM imputation**. In addition, the application is also designed to perform a **naive imputation** method.

For the case of genotype bi-allelic loci, such as most SNP data in VCF files, a vector of genotypes representing all the possible states in a variant calling file can be easily defined by using the IUPAC nucleotide codes:

- **A** for adenine/adenine homozygotes
- **C** for cysteine/cysteine homozygotes
- **G** for guanine/guanine homozygotes
- **T** for thymine/thymine homozygotes
- **E** for adenine/guanine heterozygotes
- **K** for guanine/thymine heterozygotes
- **M** for adenine/cytosine heterozygotes
- **S** for cytosine/guanine heterozygotes
- **W** for adenine/thymine heterozygotes
- **Y** for cytosine/thymine heterozygotes

- **N** in the case of missing data.

Naive imputation

Naive imputation is a frequentist procedure that determines which is the most frequent genotype in the affected variant and assigns the obtained genotype to the missing variant data.

SOM imputation

Kohonen's self-organizing map (SOM; Kohonen, 2001), is a commonly used method for data clustering and visualization that fits to the analysis of genomic sequence datasets. SOM performs an unsupervised clustering process that spatially organizes the patterns found in a dataset considering only their homology, without knowing the class to which they belong. SOM is able to conduct exploratory data analysis and visualization without the need for calibration or classification of the information to be processed. In the context of genotype data, SOM orders spatially the vectorized genotypes on a two-dimensional (2D) map, providing the similarity or dissimilarity with all other groups of sequences and the clustering of the input into groups.

SOM networks have two layers of neurons with connections between them: n neurons in the input layer and m neurons in the output layer, with n << m and n*m connections. The data represent information obtained from several dimensions (2 in our case). No supervision is required to perform the classification.

The SOM imputation performed by gtImputation has two processes (Figure 1) that store the required information in three SQL tables (Box 1):

1. **Process 1**: Creation a genotype database from the VCF or tabular format file
2. **Process 2**: Run of the imputation process itself using the genotype database

Process 1. Fetching the information from the VCF or tabular format file and creation of SQL databases. The linkage disequilibrium between each pair of SNPs according to Ragsdale & Gravel (2020) and sample kinship according to Goudet et al. (2018) are calculated and saved in a Sqlite database (Box 2). This process should be run once per VCF file.

Process 2. Imputation of the missing data from the VCF file whose genomic SQL database was built in Process 1 (Box 3). In order to create SOM objects, gtImputation uses MiniSom software (Vittigli, 2018). Process 2 can be run several times by varying the input parameters.
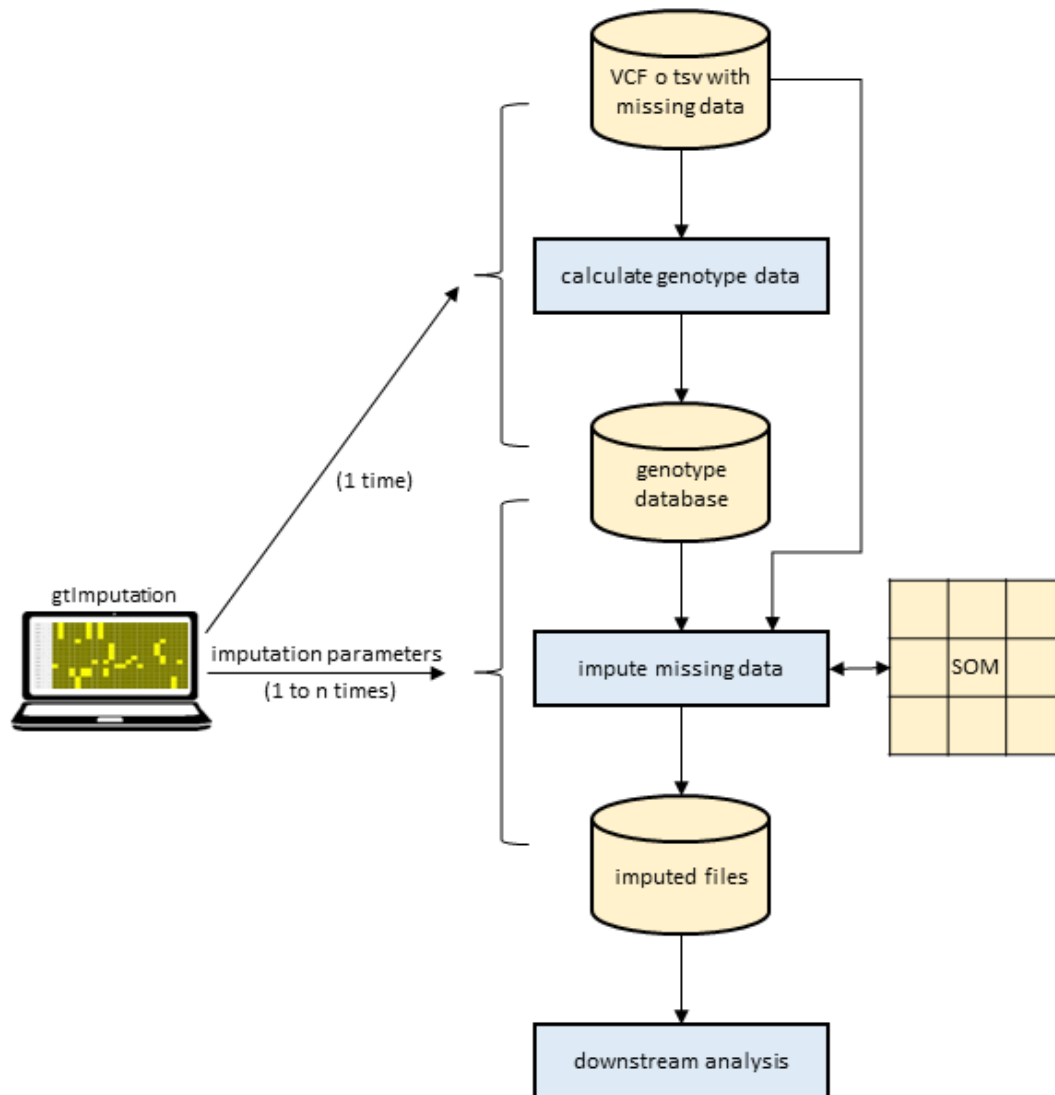
Figure. 1. Flow-chart of gtImputation.

The user can modify the following MiniSom input parameters:

- x and y dimensions of the SOM.
- sigma: spread of the neighborhood function, needs to be adequate to the dimensions of the map.
- initial learning rate.
- maximum number of iterations for the training.

The other MiniSOM parameters are set by default or by running the process.

Other mandatory input parameters of gtImputation related with the selection of SNP to construct the vector sequences that the SOM will consider to assign the samples to their corresponding neurons are the following:

- a minimum squared correlation (r^2) threshold between SNP.
- a maximum number of SNP.

Notice that while running the SOM, in case the maximum number of SNP is lower than the number of SNP obtained by applying the r^2 threshold, the maximum number of SNP with the higher r^2 are considered. To fix a number of SNP the r^2 should be fixed to 0.

Once the SNP vector sequences are constructed, they are converted to 0-1 vectors. The traditional SOM works with a continuous numerical input space and Euclidean distances. In order to work with nucleotide sequences, gtImputation made a conversion to numerical values using an array of 10 positions such that the first position corresponds to the character 'A', the second to the character 'C' and so on up to the tenth which corresponds to the character 'Y'. Each coded genotype assigns a 1 to the element of the array corresponding to its character and 0 to the rest. In the case of missing data, all the elements of the array have a value of 0 (Figure 2).
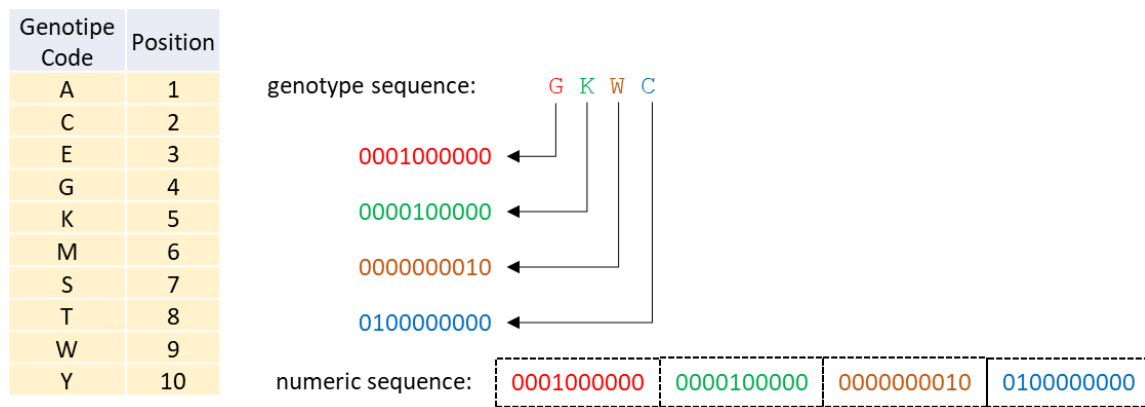
Genotipe Code	Position
A	1
C	2
E	3
G	4
K	5
M	6
S	7
T	8
W	9
Y	10

genotype sequence: G K W C

0001000000

0000100000

0000000010

0100000000

numeric sequence: | 0001000000 | 0000100000 | 0000000010 | 0100000000 |

Figure. 2. From a genotype sequence to its numeric sequence.

Once the SOM is created, it is trained with the numeric vectors of each sample and each sample is assigned to a specific neuron (Figure 3). The neuron that corresponds to a sample with missing data is identified and there are two possibilities to impute the genotype:

- imputing the most frequent genotype of the samples found in the neuron hosting the sample with missing data.
- imputing the genotype of the sample with the closest kinship found in the neuron.

For further information refer to the manuscript:

Mora-Márquez F, Nuño J. C., Soto A & López de Hereda U (in review). Missing genotype imputation in non-model species using self-organizing maps.
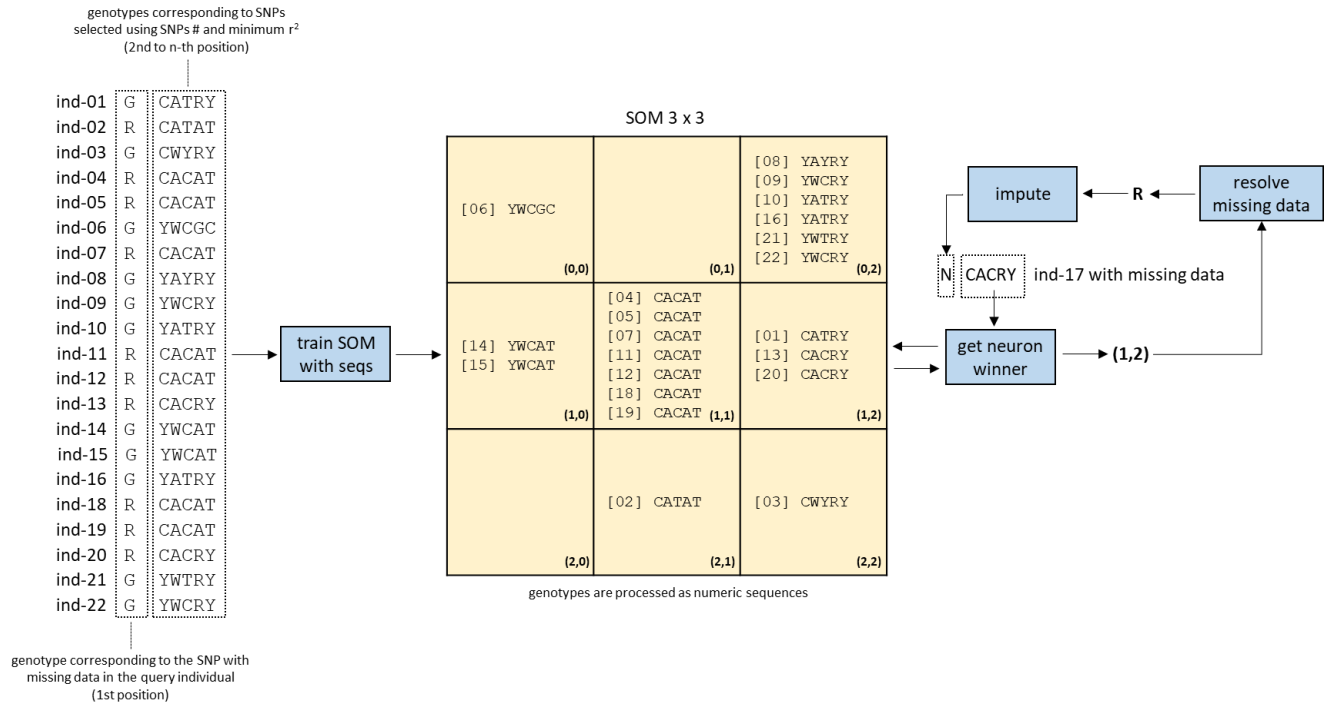
Figure. 3. SOM training and imputation process in gtImputation..

Box 1: Description of the three SQL tables used by gtImputation to store the information fetched from the VCF file in Process 1 and employed in Process 2

```
##################
# Table vcf_snps #
##################

Column                  Type    Comment
===================---====    =======================================================================
variant_id              TEXT    identification of the variant (fields CHROM-POS of VCF file)
ref                     TEXT    reference allele (field REF of VCF file)
alt                     TEXT    alternative allele(s) (field ALT of VCF file)
sample_gt_list          TEXT    comma-separated of sample (individual) genotypes in the variant
sample_withmd_list      TEXT    list of samples (individuals) with missing data


#####################
# Table vcf_kinship #
#####################

Column            Type      Comment
============---=======    ===================================================================
individual_i      INTEGER   identification of an individual
individual_j      TEXT      identification of another individual
ru                REAL      unweighted average estimator of kinship (formula 4, Goudet, 2018)


###################################
# Table vcf_linkage_disequilibrium #
###################################

Column                  Type    Comment
===================     ====    ================================================================
snp_id_1                TEXT    identification of a SNP with missing data (fields CHROM-POS)
snp_id_2                TEXT    identification of one of the other SNP (fields CHROM-POS)
dhat                    REAL    unbiased estimator for the covariance of alleles co-occurring on a
                                haplotype
r_2                     REAL    squared correlation
sample_withmd_list_2    TEXT    list of samples (individuals) with missing data


Note:

A sample (individual) is identified by the order in which it appears in the fields SAMPLE: the first
sample is 0; the second one, 1; and so on.
```

Box 2: Pseudocode of the creation of a genotype database from the information fetched from a VCF file.

```
##################################################
# Creation a genotype database from the VCF file #
##################################################

function calculate_genotype_data (database, vcf_file)

    initialize sample_list and sample_number
    initialize kinship_dictionary (structure: {snp_i: {snp_j: {summation, l}})

    do connection to database

    drop the table vcf_snps (if it exists)
    create the table vcf_snps

    drop the table vcf_linkage_disequilibrium (if it exists)
    create the table vcf_linkage_disequilibrium

    drop the table vcf_kinship (if it exists)
    create the table vcf_kinship

    open the vcf_file
    read the first record of vcf_file
    while there are records in vcf_file

        while there are record in vcf_file and they corresponding to metadata
            read the next record of vcf_file

        if the record corresponding to the column description
            build sample_list and sample_dictionary
            calculate samples_number
            set 0 in the summation and l values in kinship_dictionary
                    (used in the calculation of the unweighted average estimator of kinship)
            read the next record of vcf_file

        while there are records in vcf_file and they corresponding to variants
            variant_id = f'{field CHROM}-{field CHROM}'
            get the reference allele (field REF) and alternative alleles (field ALT)
            if the variant has more than one alternative allele
                end the process with error
            get the genotype_position (subfield GT in the field FORMAT)
            build sample_genotypes_list considering genotype_position in fields SAMPLE
            build samples_with_missing_data_list
            update kinship_dictionary adding this calculation in the summation for each
                    samples i and j:
                        (Xi - 2 * p) * (Xj - 2 * p) / (2 * p * (1 - p))
                    where Xi and Xj are the dosage of reference allele for samples i and j
                    respectively and p is the frequency of reference allele in the current variant
                    (do not-update when there is missing data in samples i or j, or p value
                    is 0 or 1)
            update kinship_dictionary adding 1 in l for samples i and j
                    (do not-update when there is missing data in samples i or j, or p value
                    is 0 or 1)
            insert SNP data into table vcf_snps if there is more than one genotype
            read next record of vcf_file

    close vcf_file

    for each samples i and j:
        calculate the unweighted average estimator as summation / l
        insert kinship data into the table vcf_kinship from kinship_dictionary

    get snp_ids_list from the table vcf_snps
    get snp_with_missing_data_ids_list from the table vcf_snps
```

```
    for each snp_id in snp_with:missing_data_ids_list:
        calculate_snp_linkage_disequilibrium (connection, sample_number, snp_id, snp_ids_list)

    save changes into database
    close connection to database

    close vcf_file
```

```
###################################################
# Calculation the linkage disequilibrium of a SNP #
###################################################


function calculate_snp_linkage_disequilibrium (connection, sample_number, snp_id_1, snp_id_2_list)

    get sample_genotypes_1_list of snp_id_1 from the table vcf_snps

    for each snp_id_2 in snp_id_2_list (except snp_id_1):

        get sample_genotypes_2_list of snp_id_2 from the table vcf_snps

        calculate the observed genotype counts and allele frequencies considering:

                ri: reference allele of SNPi
                ai: alternative allele of SNPi
                ni: count of observed genotype pair i
                n: summation of ni

                    r2/r2 r2/a2 a2/a2
                  +-----------------
            r1/r1 |  n1    n2     n3
            r1/a1 |  n4    n5     n6
            a1/a1 |  n7    n8     n9

        calculate the unbiased estimator for the covariance of alleles co-occurring on a haplotype
                according to:
                dhat = ((n1 + n2/2 + n4/2 + n5/4) * (n5/4 + n6/2 + n8/2 + n9) –
                    (n2/2 + n3 + n5/4 + n6/2) * (n4/2 + n5/4 + n7 + n8/2)) / (n * (n - 1))

        calculate the squared correlation according to:

                rfi: reference allele frequency of SNP i
                afi: alternative allele frequency of SNP i

                r_2 = (dhat ** 2) / (rf1 * af1 * rf2 * af2)

        insert linkage disequilibrium data into the table vcf_linkage_disequilibrium
```

Box 3: Pseudocode of the missing data imputation using Self-Organizing Maps and the genotype database created previously with the input VCF file

```
##########################################################################
# Imputation of genotypes with missing data in a VCF file using Self-Organizing Maps #
##########################################################################

function impute_md_som (database, input_vcf_file, imputed_vcf_file, minimum_r2, snps_num, xdim, ydim,
                        sigma, learning_rate, num_iteration, genotype_imputation_method)

    initialize sample_list and sample_number

    do connection to database

    get kinship_dictionary (structure: {snp_i: {snp_j: {summation, l}}) from table vcf_kinship

    get snp_ids_list with linkage disequilibrium data from the table vcf_linkage_disequilibrium

    open input_vcf_file
    open imputed_vcf_file

    read the first record of input_vcf_file
    while there are record in input_vcf_file

        while there are records in input_vcf_file and they corresponding to metadata
            read the next record of input_vcf_file
            write the metadata record in imputed_vcf_file

        if the record corresponding to the column description
            build sample_id_list
            calculate the samples_number
            write column description record in imputed_vcf_file
            read the next record of input_vcf_file

        while there are records in input_vcf_file and they corresponding to variants
            variant_id = f'{field CHROM}-{field CHROM}'
            get the reference allele (field REF) and alternative alleles (field ALT)
            if the variant has more than one alternative allele
                end the process with error
            get the genotype_position (subfield GT in the field FORMAT)
            build sample_genotypes_list considering genotype_position in fields SAMPLE
            if variant_id in snp_ids_list:
                get samples_with_missing_data_list of the variant from table vcf_snps
                get best_snps_ids_list with SNP ids with the highest r^2 values regarding variant_id
                selected_snp_ids_list = [variant_id] + best_snps_ids_list
                build sample_sequence_list corresponding to selected_snp_ids_list
                build sample_numeric_haplotypes_list from sample_sequence_list
                calculate most_frequent_genotype
                if there is one genotype:
                    set most_frequent_genotype as genotype of samples with missing data
                else:
                    build the input data to the SOM algorithm
                    build the training data corresponding to samples without missing data
                        from the input data
                    build the test data corresponding to samples with missing data from the input data
                    create a new SOM xdim x ydim instance with sigma, learning_rate, num_iteration
                        (default values for all other parameters)
                    initialize the weights to span the first two principal components
                    train the SOM
                    for each sample_with_missing_data in samples_with_missing_data_list:
                        get winning_neuron for samples_with_missing_data
                        if genotype_imputation_method is the most frequent genotype in the neuron:
                            get samples_in_neuron_list in wining_neuron
                            calculate most_frequent_genotype_in_neuron from samples_in_neuron_list
                            set most_frequent_genotype_in_neuron in genotype of
```

```
                              sample_with_missing_data
                elif genotype_imputation_method is the genotype of closest kinship individual:
                    get the most_related_sample_id from kinship_dictionary
                    set genotype most_related_sample_id of in genotype of
                              sample_with_missing_data
                build the genotype text after imputation
            rebuild the sample genotype data list and their corresponding record data
            write the variant record in imputed_vcf_file
            read the next record of input_vcf_file

    close connection to database

    close input_vcf_file
    close imputed_vcf_file
```

Limitations

Multiallelic loci and other types of variants are not considered.
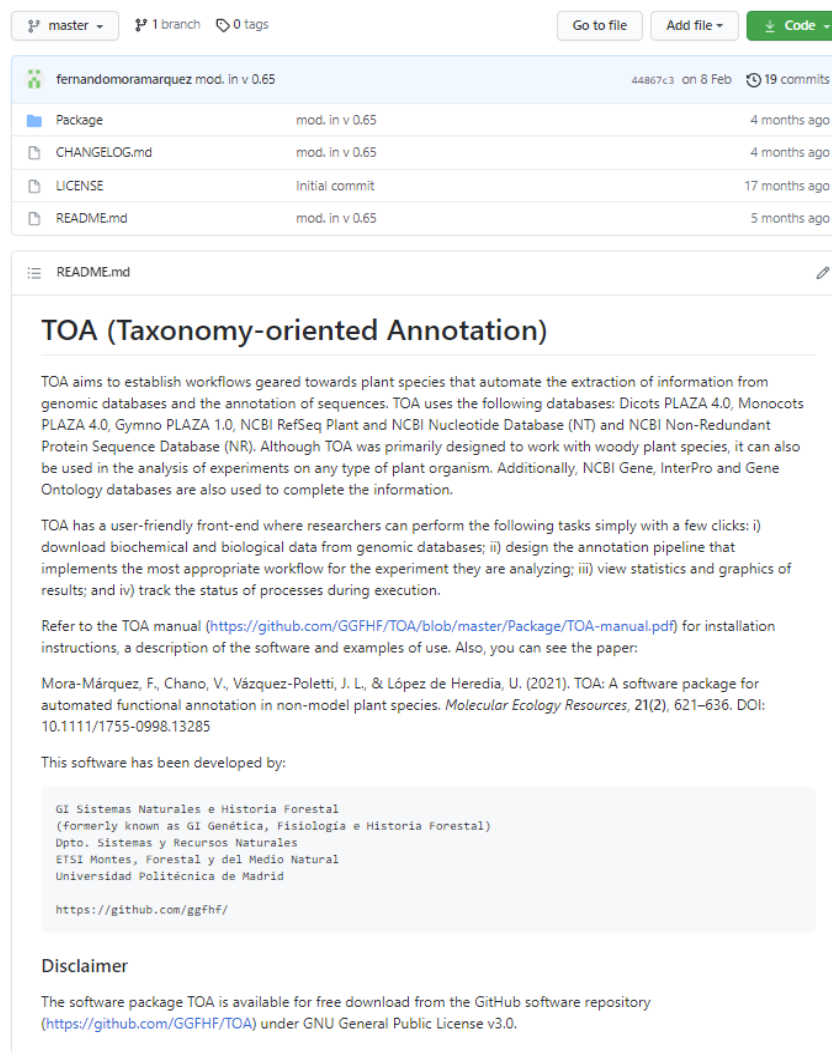
Samples should belong to a single population/species.

Limitations

Installation

gtImputation installation

gtImputation was programmed in Python 3 and it generates dynamic Bash scripts to perform annotation pipelines. gtImputation runs in any computer with Linux, macOS or Windows with WSL (Windows Subsystem for Linux).

gtImputation is available from the GitHub software repository in the URL https://github.com/GGFHF/gtImputation/, and it is distributed under GNU General Public License Version 3 (Figure 2).

Figure. 2. gtImputation home at GitHub software repository.

To download gtImputation, click in *Code* and in the pup-up window click in *Download ZIP*.

To install gtImputation on Linux and macOS, simply decompress the gtImputation-main.zip into a directory, typing the following command in a terminal window:

```
$ unzip gtImputation-master.zip
```

Then, the execution permissions of the programs must be set by using this command:

```
$ chmod u+x *.py
```

Additional infrastructure software installation

Python 3, version 3.9 or higher, is necessary for a correct functioning of gtImputation. If Python 3 is not installed in your computer, you can download it from the official website (https://www.python.org/), or use one of the several distributions that include Python along with other software packages for standard bioinformatic analysis such as Anaconda (https://www.anaconda.com/products/distribution/).

To work properly, gtImputation needs the following Python modules:

- PyQt5 (https://www.riverbankcomputing.com/static/Docs/PyQt5/), a Python interface for QT software package.
- …

Next, we present how to install this additional software in two example environments: a) an Ubuntu Linux 22.04 where Python3 is pre-installed in the OS; b) a macOS 12.6.1 where Python is installed using Anaconda; c) a Windows 10 using WSL and Anaconda.

Ubuntu 22.04 LTS (Jammy Jellyfish)

On Ubuntu, Python 3 is pre-installed. To install PyQt5, if necessary, open a terminal window and type the following commands:

```
$ sudo pip3 install pyqt5
```

```
$ [sudo apt install libxcb-xinerama0]
```

macOS 12.6.1 (Monterey)

Since Python 3 is not installed, the Anaconda distribution is a solution. First, install Homebrew and wget command, if necessary, typing the following commands in the terminal window:

```
$ /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

```
$ brew install wget
```

Now download the Anaconda software file, e.g. the version 2022.10, typing the command:

$ wget https://repo.anaconda.com/archive/Anaconda3-2022.10-MacOSX-x86_64.sh

And provide execution permission to this file and run it typing the commands:

$ chmod u+x Anaconda3-2022.10-MacOSX-x86_64.sh

$ ./Anaconda3-2022.10-MacOSX-x86_64.sh

During the Anaconda installation, read the Anaconda End User License Agreement, accept the license terms and indicate the location where Anaconda will be installed.

The appearance of the file *.zhsrc* must be similar to the file shown in the Figure x.

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/Users/user/anaconda3/bin/conda' 'shell.zsh' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/Users/user/anaconda3/etc/profile.d/conda.sh" ]; then
        . "/Users/user/anaconda3/etc/profile.d/conda.sh"
    else
        export PATH="/Users/user/anaconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

Figure. X. An example of the file macOS *.zshrc*.

It is also possible to install the Anaconda distribution using a graphical installer available at the url https://www.anaconda.com/products/distribution/.

PyQt5 is installed by default in Anaconda distribution.

Microsoft Windows 10 (64-bits)

gtImputation uses the Windows Subsystem for Linux (WSL) and Ubuntu to run some scripts coded in Bash. Therefore, these two software applications have to be installed before using gtImputation. They can be installed from the Windows Store. For further clarification about WSL, you can see the url https://learn.microsoft.com/en-us/windows/wsl/.

In order to have Python 3 in a Windows computer, we can install the Anaconda distribution. This distribution also installs the PyQt5. There is a Anaconda graphical installer for Windows in the url https://www.anaconda.com/products/distribution/.

Next, you must review the window "Edit environment variable" in "System Properties" and verify that the following directories are declared as PATH variables:

- *Anaconda3_path*
- *Anaconda3_path*\Library\mingw-w64\bin
- *Anaconda3_path*\Library\usr\bin
- *Anaconda3_path*\Library\bin
- *Anaconda3_path*\Scripts

Anaconda3_path is the directory where Anaconda3 is installed (the default directory proposed in the installation was C:\ProgramData\Anaconda3, but it could be changed).

You can see below an example of this windows:
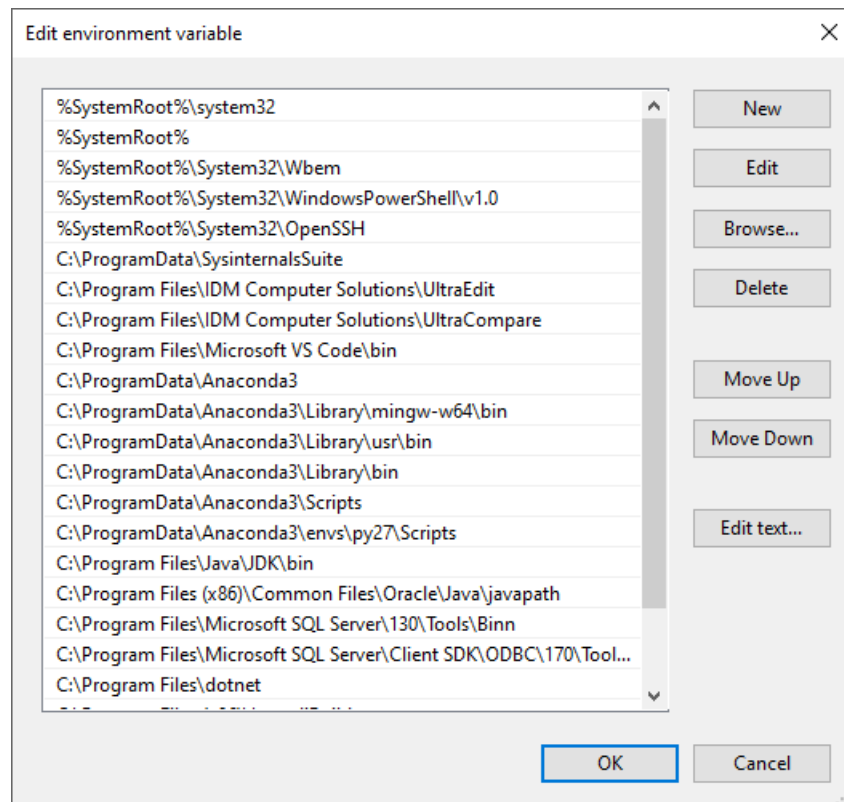
Figure. 2. Environment variable window on Windows 10.

Starting gtImputation

On Ubuntu or macOS computers, gtImputation starts typing the following command in a terminal window in the directory where the package of gtImputation is downloaded:

$ ./gtImputation.py

On Windows computer, the command to start gtImputation is:

> puthon gtImputation.py

Initial appearance of gtImputation at application startup is shown in Figure 3.
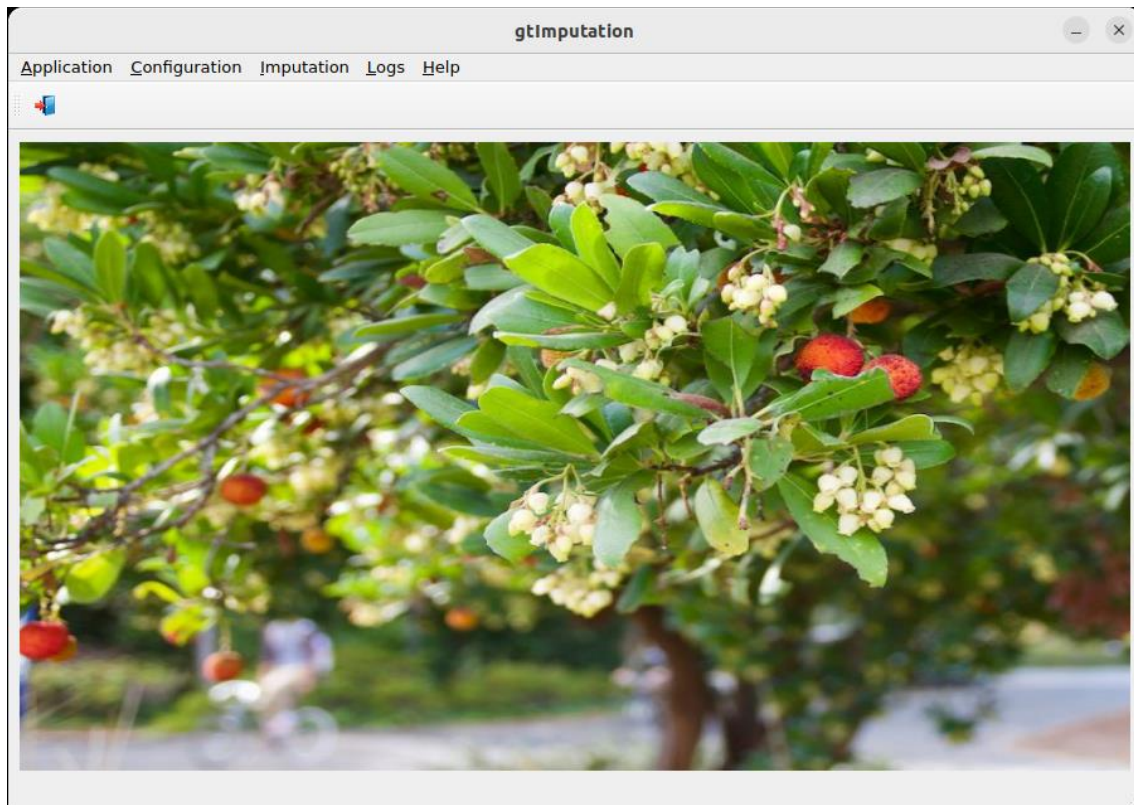
Figure. 3. Aspect of the gtImputation interface at startup.

First steps

gtImputation menus

gtImputation is structured in several menus:

System

Just to exit the application.

Configuration

This menu contains all the items related to:

- Recreate gtImputation config file
- View gtImputation config file
- Install Bioinfo software

Imputation menu

The options included here allow to run scripts to perform imputation process:

- Naive imputation
  - Run imputation process
  - Review imputation process
- SOM imputation
  - Build a genotype database
  - Run imputation process
  - Review imputation process

Logs menu

This menu allows the access to the application logs:

- View submission logs
- View result logs

Help menu

It contains the documentation of the application.

Configuring the gtImputation environment

When gtImputation starts for the first time it is required to configure the gtImputation environment. To do so, we select the menu item with the following path:

Main menu > Configuration > Recreate gtImputation config file

The Figure 4 shows the window corresponding to this menu item. Default values are presented for *Miniconda directory*, *Genotype database directory* and *Result directory*. If necessary, modify them and press the button *[Execute]*.
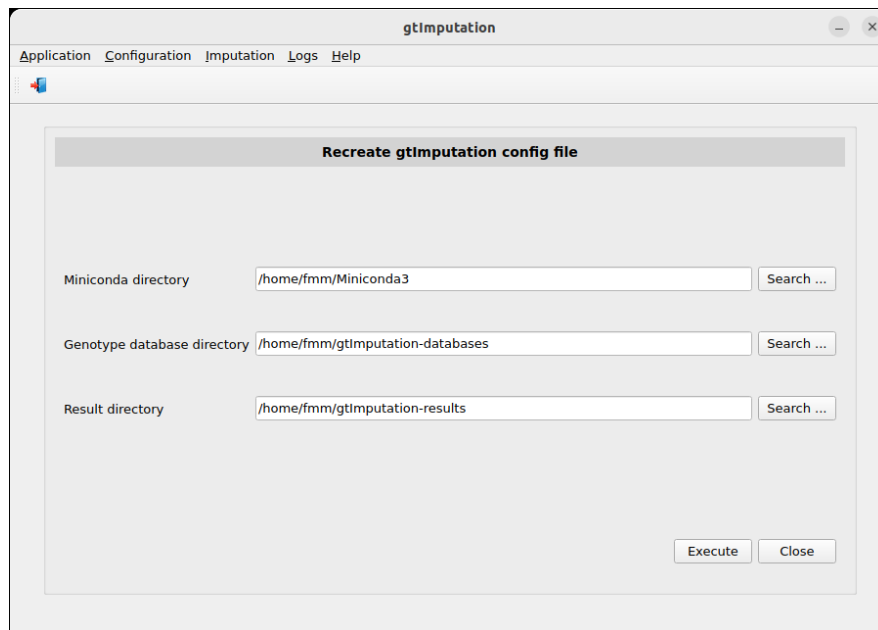
Figure. 4. gtImputation window corresponding to the menu item *Recreate gtImputation config file*.

Installing infrastructure software

gtImputation needs the following software:

- MiniSom: a minimalistic and NumPy-based implementation of the Self Organizing Map (Vittigli, 2018).

The installation of this software is automatic when Miniconda is installed Then, install Miniconda·selecting the menu item with this path:

Main menu > Configuration > Bioinfo software installation > Miniconda3 and additional infrastructure software

A new window is shown (Figure 5).
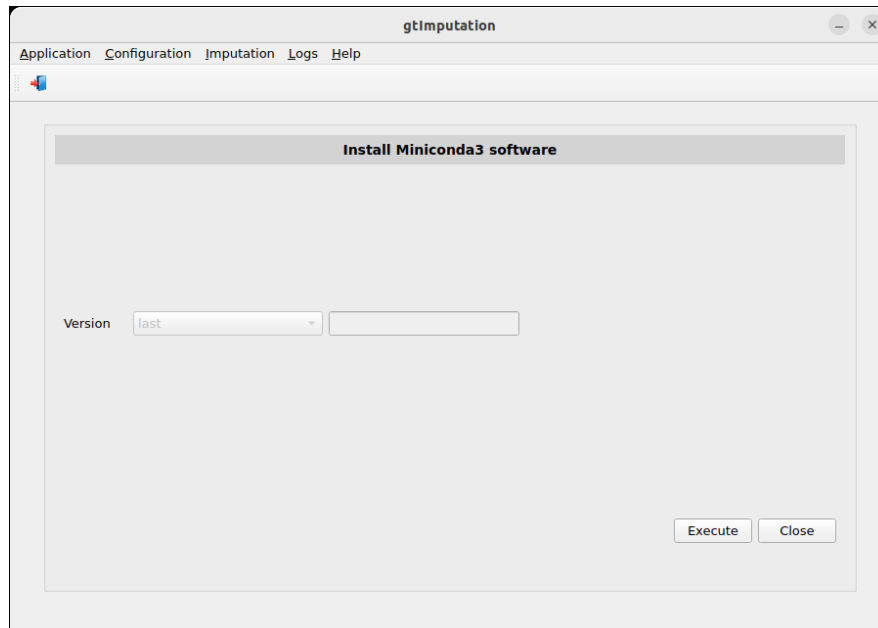
Figure.5. gtImputation window corresponding to the menu item *Miniconda3 and adiccional infrastructure software*.

Press the bottom *[Execute]*. A pop-up window will display the submission log (Figure 6).
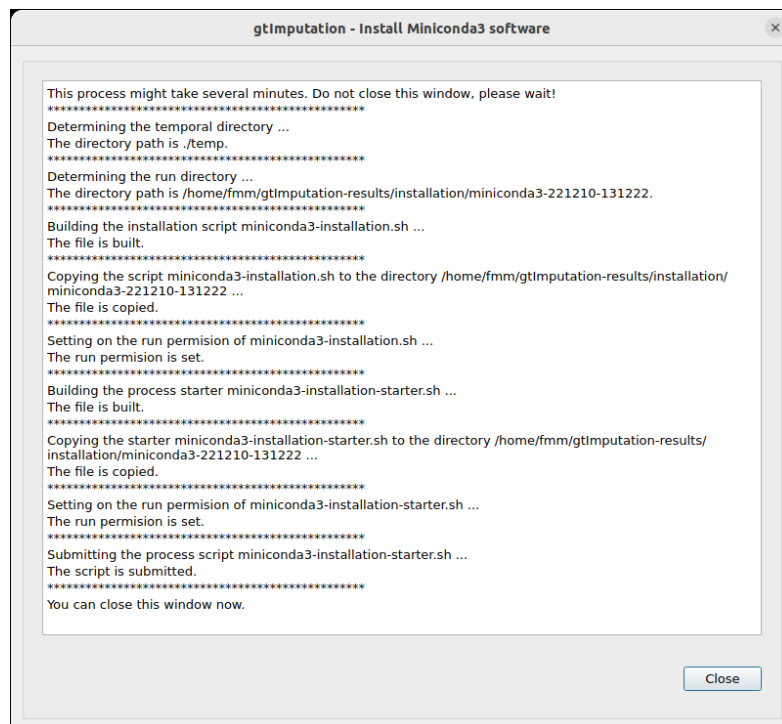
Figure.6. The log dialog showing the submission log of Miniconda3 installation process.

To view the process log during and after the run, select the menu item with this path:

Main menu > Logs > Result logs

In the raised window (Figure 7), select **installation** in the *Process type* combo-box*.*
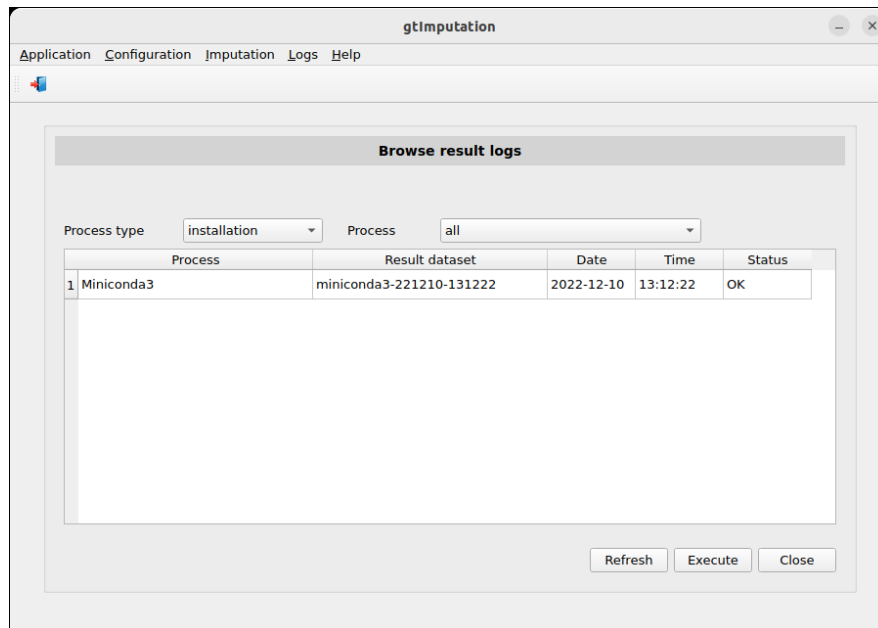
Figure. 7. gtImputation window corresponding to the menu item *Logs - Result logs* after ended the Miniconda installation.

Since a single installation process was performed, the process **miniconda3-20221210-131222** corresponding to the last (and unique) Miniconda3 installation. Double click on it, or click on it and on the button Execute. Then, a pop-up window appears with its corresponding log (Figure 8).

All the process logs have:

- A header with the time when it started.
- At the bottom, a summary with the status (OK, if all the programs have ended without errors; WRONG, otherwise), the end time, and the duration of the script run.

In the log dialog, there is a button to refresh the run status. If the process has not ended, click on it and the log will be updated.
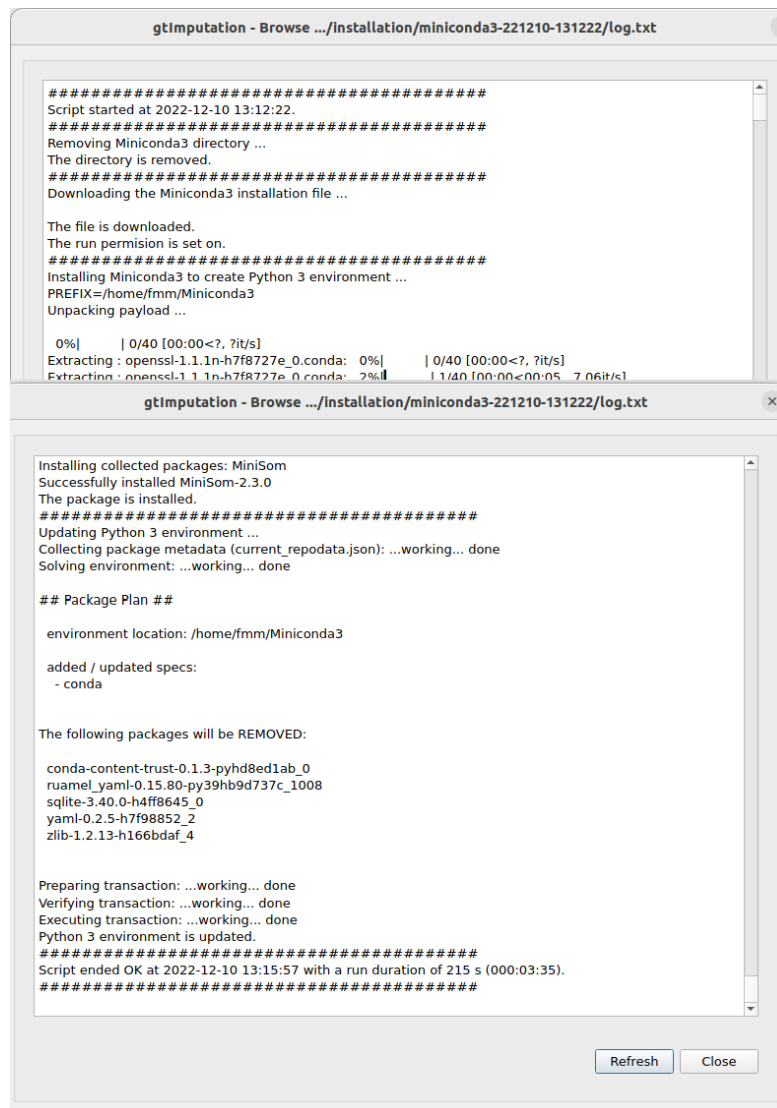
Figure. 8. Begin and end of the log installation of Miniconda3 software.

Consulting submitted processes and troubleshooting

The correct operability of the submitted processes is controlled by logs similar to the Miniconda3 installation (see Figure 6). So, the user can monitor the performance of the process at any time and detect problems. Please, confirm that each process is ended before submitting other one.

A step-by-step example

Here is a brief tutorial on how to use gtImputation.

SOM imputation

Build a genotype database

A genotype database holds sample genotypes of SNPs, linkage disequilibrium between each pair of SNPs and sample kinship got of a VCF file. These data are used by SOM imputation processes. First, select menu item with this path:

Main menu > Imputation > SOM imputation > Buid a genotype database

Then, gtImputation presents a window (Figure 9) where you indicate the *VCF file* typing its path or selecting it with the button *Search*. In this example, the file **variants-selected.vcf** is used. This file is included in the subdirectory test_data of the gtImputation software package.
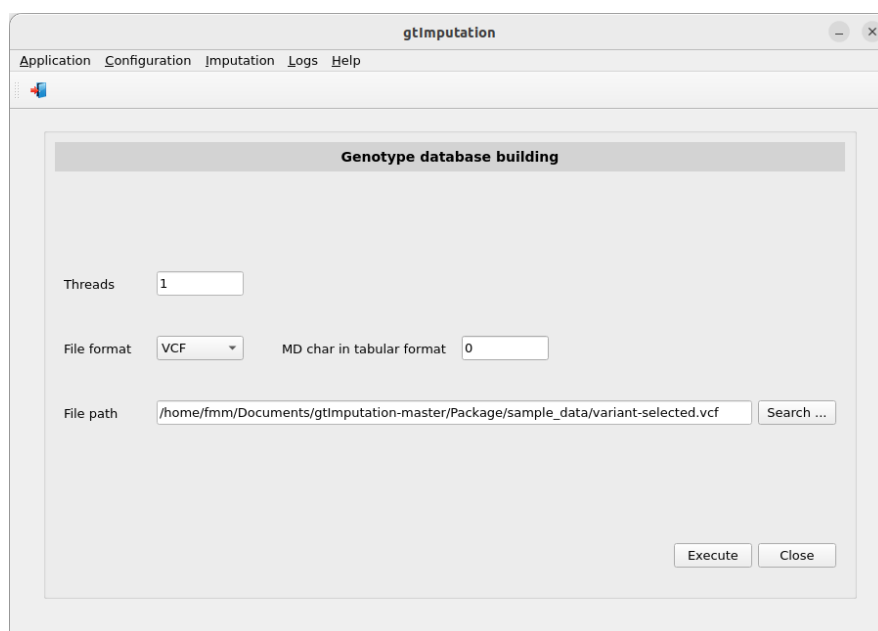
Figure. 9. gtImputation window corresponding to the menu item *Imputation > SOM imputation > Buid a genotype database*.

Once this is done, click in button *Execute*. Then a log dialog is opened with the submission information. (Figure 10).
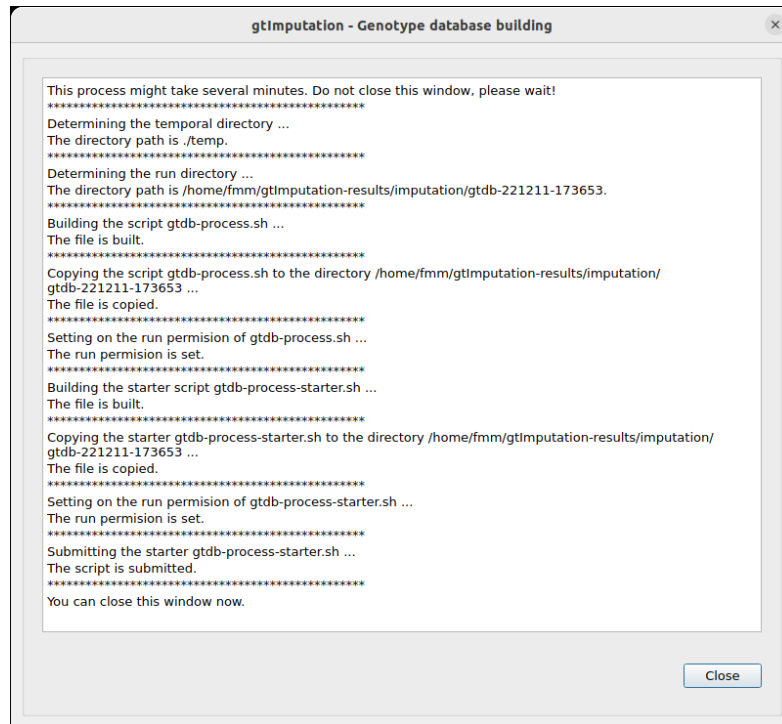
Figure. 10. The dialog showing the submission log of genotype database building.

To view the process log during and after the run, select the menu item with this path:

Main menu > Logs > Result logs

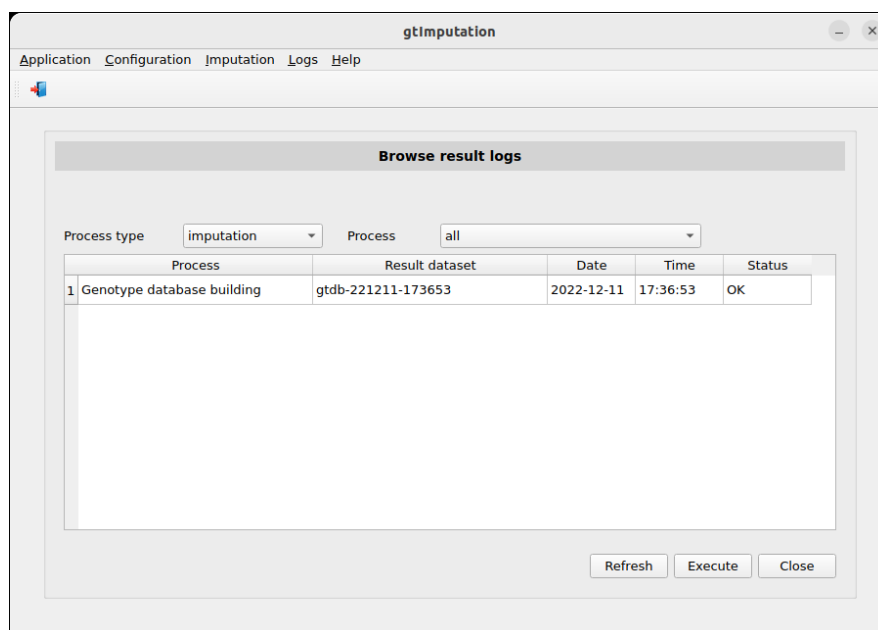In the raised window (Figure 11), select **imputation** in the *Process type* combo-box*.*

Figure. 11. gtImputation window corresponding to the menu item *Logs - Result logs* after ended the genotype database building.

Double clicking on the row corresponding to **gtdb-221211-173653** or clicking on it and on the button Execute. Then a pop-up window appears with its corresponding log (Figure 12).
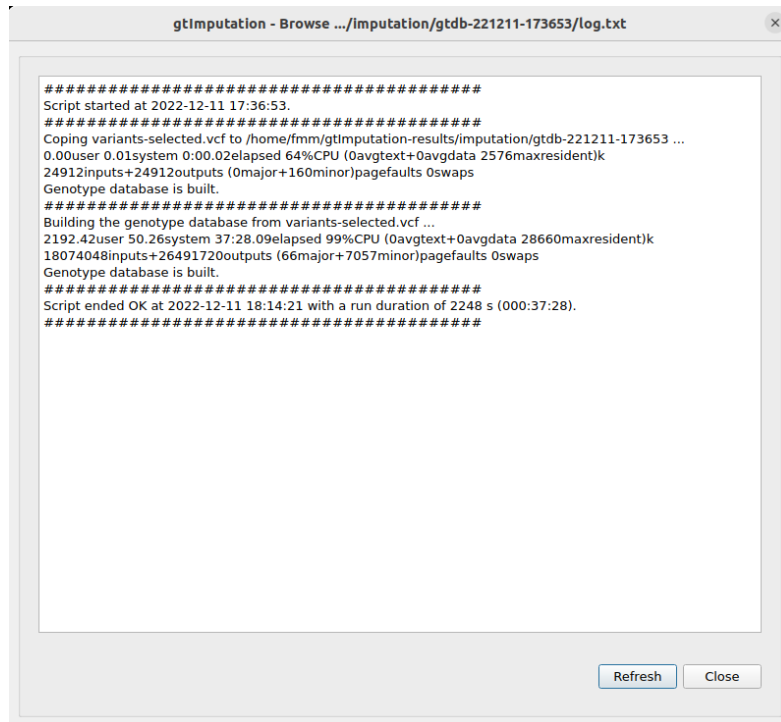
Figure. 12. The dialog showing the process log of genotype database building.

Run imputation process

Once the genotype dataset is built, you can run one or several SOM imputation processes with the appropriate parameters. To do so, select the menu item with this path:

Main menu > Imputation > SOM imputation > Run imputation process

Then, gtImputation presents a window (Figure 13) where the application shows default parameters. Select the genotype database corresponding to the VCF file you need to impute by clicking its identification in the corresponding combo-box and, below, set the appropriate parameters.

The combo box *Genotype database* is loaded with the process identifications of genotype database building ended OK. In this tutorial, we will use the database built previously. Select the identification **gtdb-221211-173653** in the combo box *Genotype database* and then set **5** in the *X dimension* and *Y dimension* of *SOM parameters*.
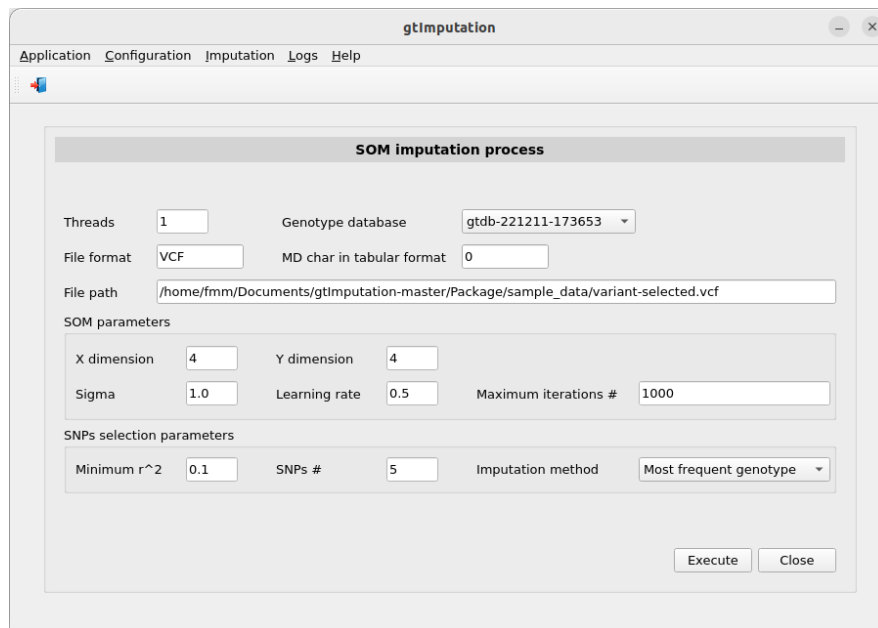
Figure. 13. gtImputation window corresponding to the menu item *Imputation > SOM imputation > Run imputation process*.

Then, click the button *Execute*. A log dialog is opened with the submission information. (Figure 14).
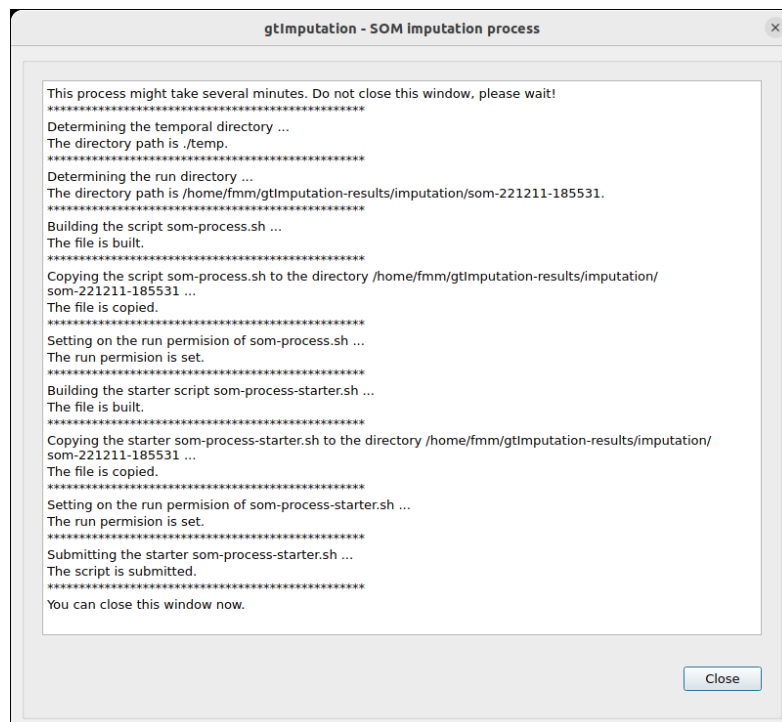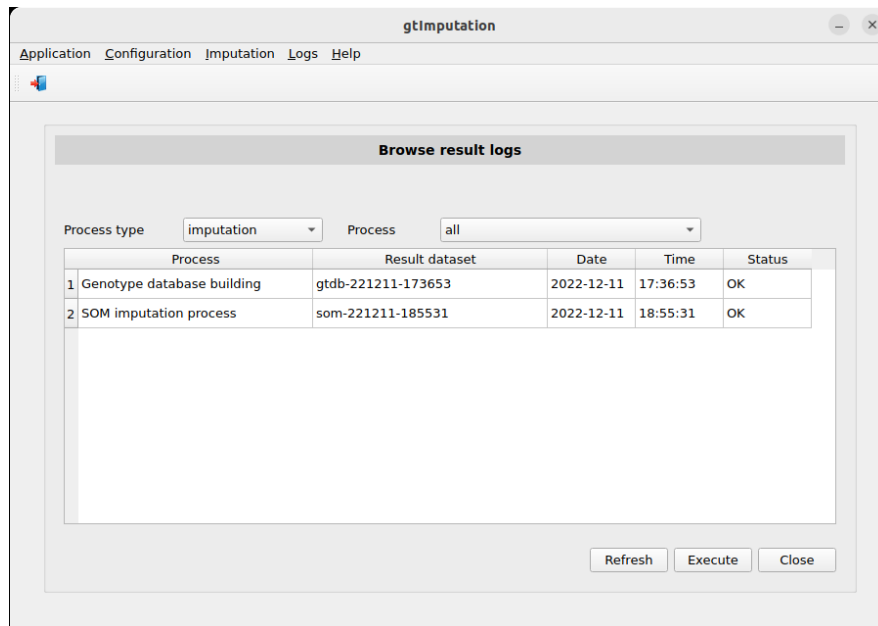

Figure. 14. The dialog showing the submission log of SOM imputation process.

To view the process log during and after the run, select the menu item with this path:

Main menu > Logs > Result logs

In the raised window (Figure 15), select **imputation** in the *Process type* combo-box.

Figure. 15. gtImputation window corresponding to the menu item *Logs - Result logs* after ended the SOM imputation process.

Double clicking on the row corresponding to **som-221211-185531** or clicking on it and on the button Execute. Then a pop-up window appears with its corresponding log (Figure 16).
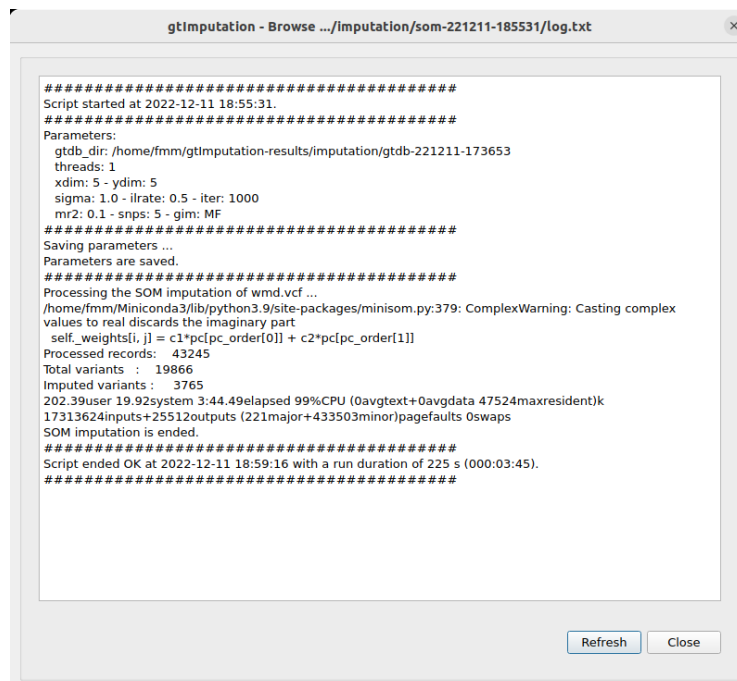

Figure. 16. The dialog showing the log of SOM imputation process.

Review imputation process

In order to review the imputation done by the SOM process, click the following menu item:

Main menu > Imputation > SOM imputation > Review imputation process

Then, gtImputation presents a window (Figure 17) where you must select the SOM process identification clicking its identification in the combo box *SOM process* which is loaded with the identifications of SOM processes ended OK. In this example, the identification of the process is **som-221211-185531**. When the identification is selected, the parameters used in the imputation process are shown.
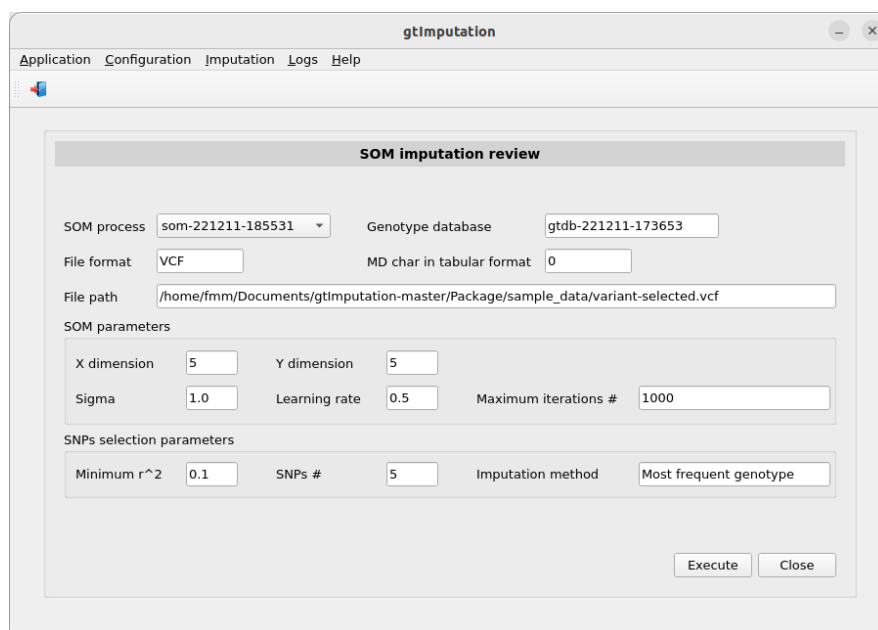
Figure. 17. gtImputation window corresponding to the menu item *Imputation > SOM imputation > Review imputation process*.

Click the execute button. Then gtImputation shows a window with the detailed imputation (Figure 18). Each row corresponds to the position of a sequence and each column represents a sample. The genotypes with missing data in the original VCF file are shown in yellow and the genotypes with data in the original VCF file in dark yellow.

If you click the button *Show map*, a summary map is shown on the left side (Figure 19) and the button changes its title to *Hide map*. Now, if you click on it, the summary map is hidden.

If you click on a sequence identification or on a position of the summary map (Figure 20), the detailed imputation will show only the positions of this sequence and the buttons *Prev sequence* and *Next sequence* will be available. These buttons allow you to browse sequence by sequence the positions of each sequence. To show all sequences, click on the button *All Sequences*
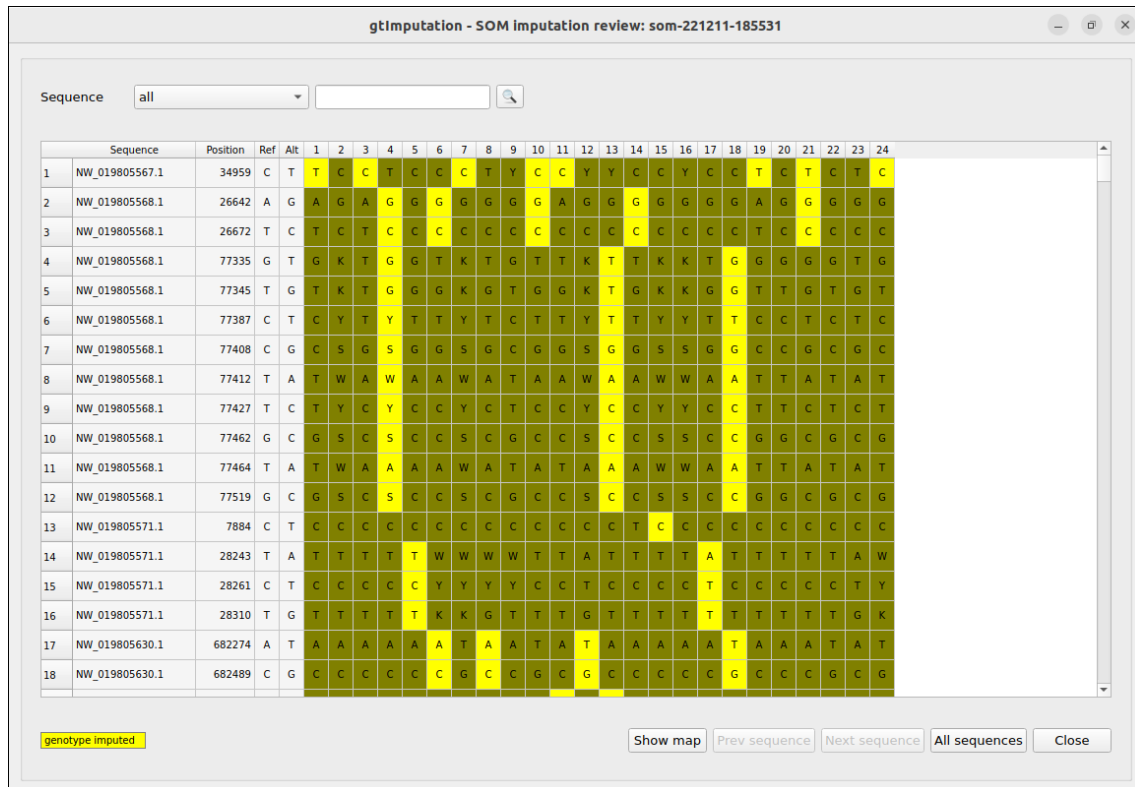
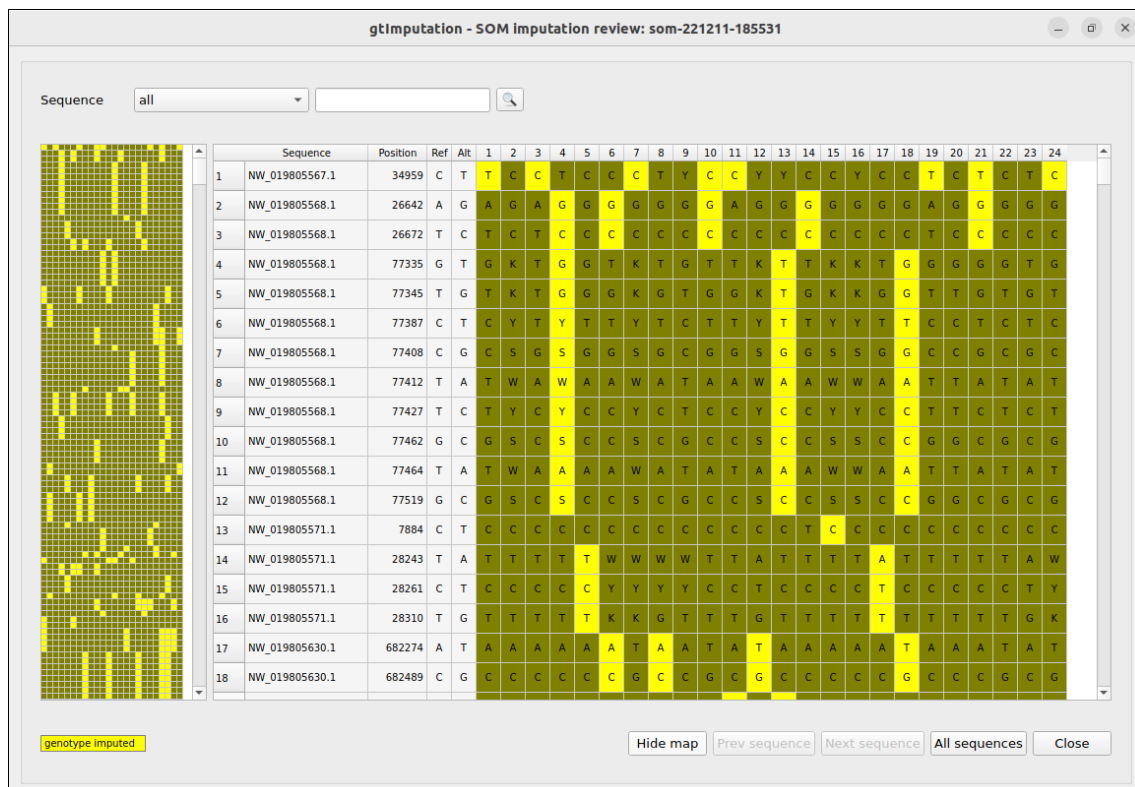**Figure. 18**. gtImputation window showing the detailed imputation.

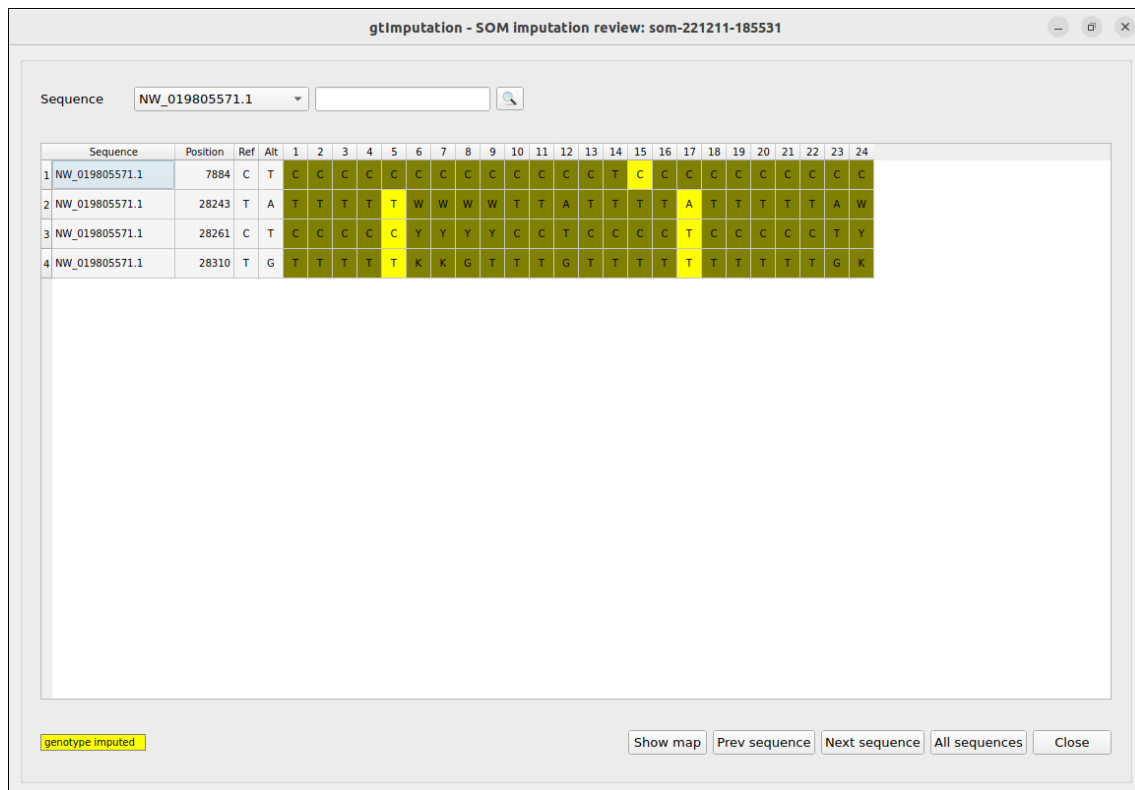**Figure. 19**. gtImputation window showing both the detailed imputation and a summary map.

Figure. 20. gtImputation window showing the positions of a sequence.

Naive imputation

Run imputation process

A naive imputation processes are run from the following menu item:

Main menu > Imputation > Naive imputation > Run imputation process

Click this menu item. gtImputation presents a window (Figure 21) where you indicate the *VCF file* typing its path or selecting it with the button *Search*. Like the SOM imputation example, the file **variants-selected.vcf** is used which is included in the subdirectory test_data of the gtImputation software package.
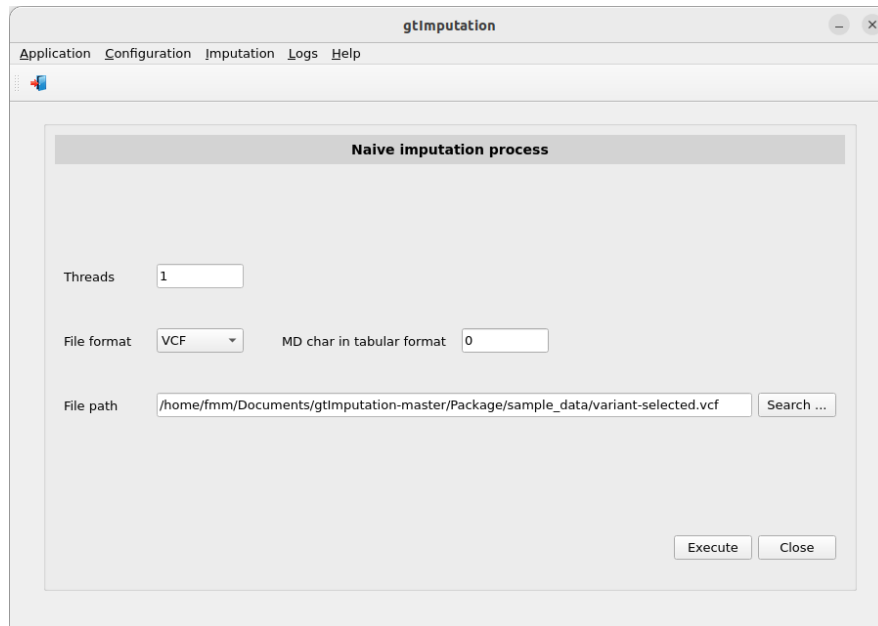
Figure. 21. gtImputation window corresponding to the menu item *Imputation > Naive imputation > Run imputation process*.

Then, click the button *Execute*. A log dialog is opened with the submission information. (Figure 22).
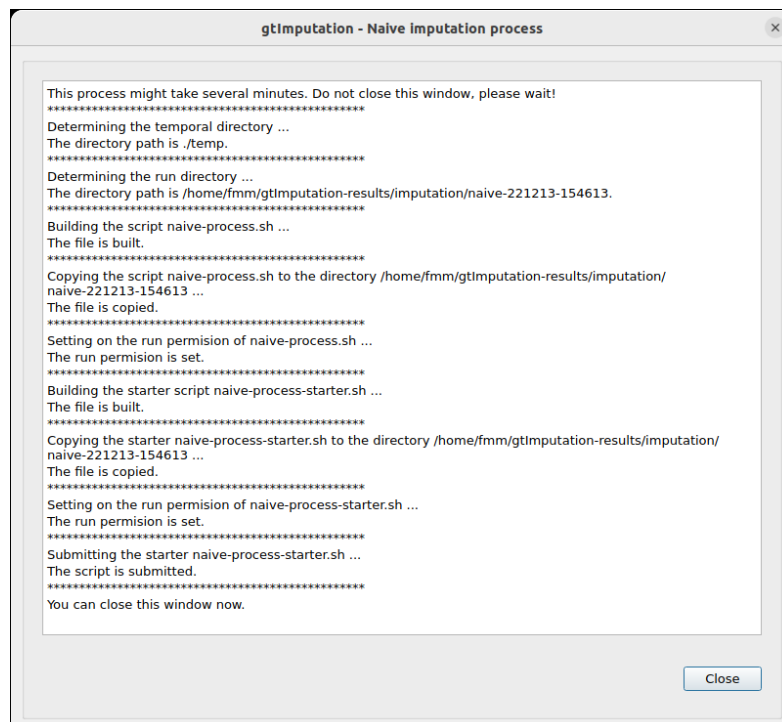

Figure. 22. The dialog showing the submission log of naive imputation process.

To view the process log during and after the run, select the menu item with this path:

Main menu > Logs > Result logs

In the raised window (Figure 23), select **imputation** in the *Process type* combo-box.
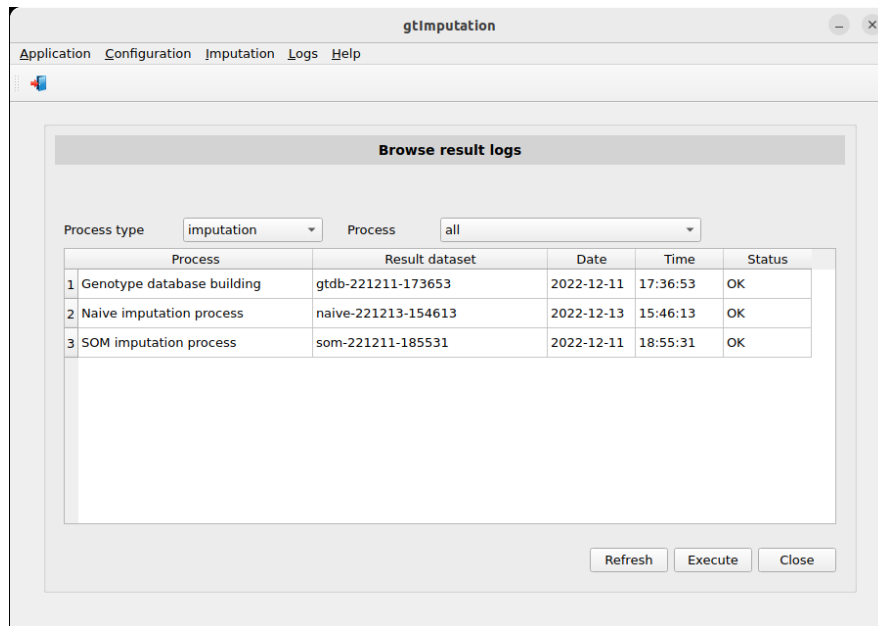

Figure. 23. gtImputation window corresponding to the menu item *Logs - Result logs* after ended the naive imputation process.

Double clicking on the row corresponding to **naive-221213-154613** or clicking on it and on the button Execute. Then a pop-up window appears with its corresponding log (Figure 24).
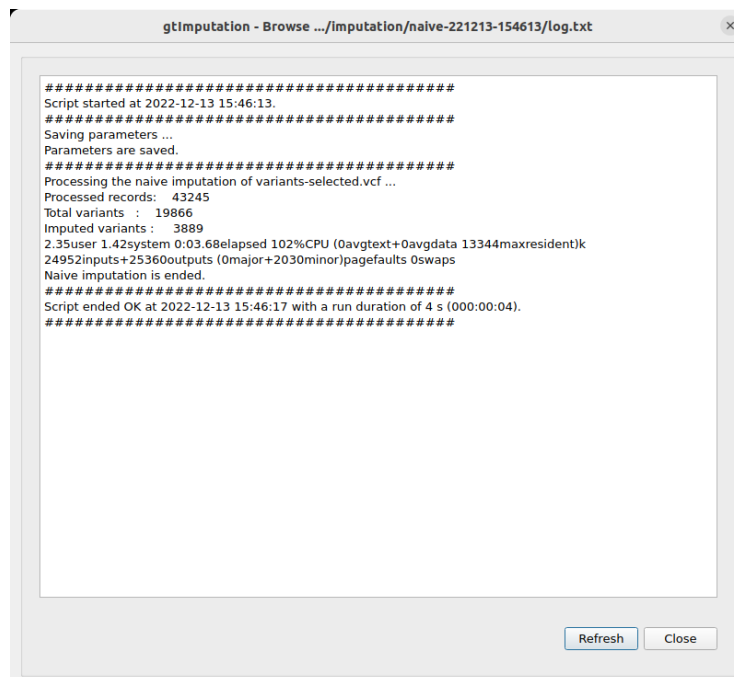

Figure. 24. The dialog showing the log of naive imputation process.

Review imputation process

The review of a naive process is done clicking the following menu item:

Main menu > Imputation > Naive imputation > Review imputation process

Then, gtImputation presents a window (see Figure 25 where you must select the naive process identification clicking its identification in the combo box *Naive process* which is loaded with the identification of the naive processes ended OK. In this example, the identification of the process is **naive-221213-154613**. When the identification is selected, the VCF file used in the imputation process is shown.
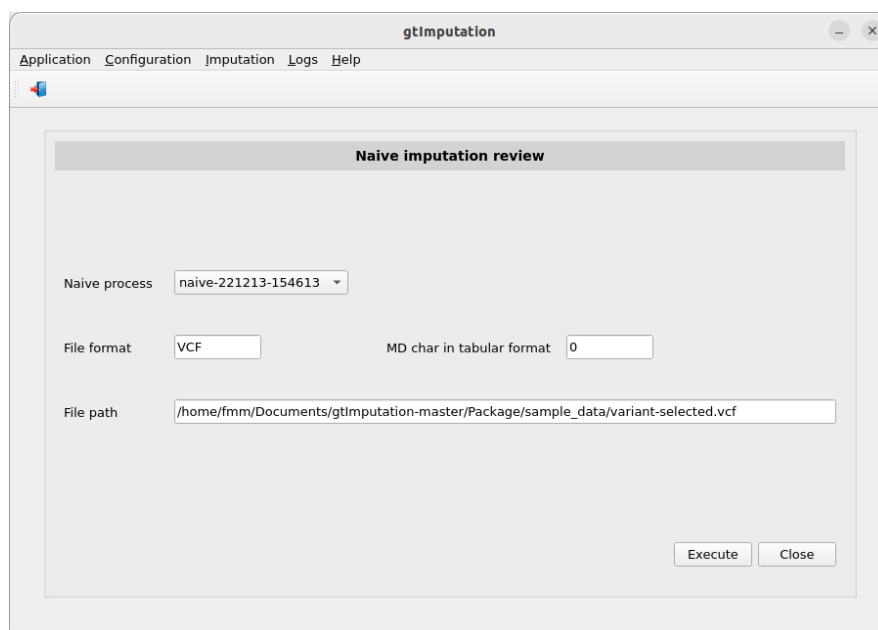

Figure. 25. gtImputation window corresponding to the menu item *Imputation > Naive imputation > Review imputation process*.

Click the execute button. Then gtImputation shows the imputation window. The review of a naive imputation process is similar to the procedure described for the SOM imputation.

Where are the imputed files?

An imputation process generates two file in the result dataset directory whose names are **imputed.vcf** and **imputed.tsv** (in tabular format) with the imputed genotypes in the missing data of the original VCF file. For instance, in this tutorial, the imputed.vcf in the example of the SOM process is in the directory path (see Figure 26):

result_dataset_directory/imputation/som-221211-185531

where *result_dataset_directory* is set in the menu item (Figure 4):

Main menu > Configuration > Recreate gtImputation config file

You can check the current value of the in the following menu item:

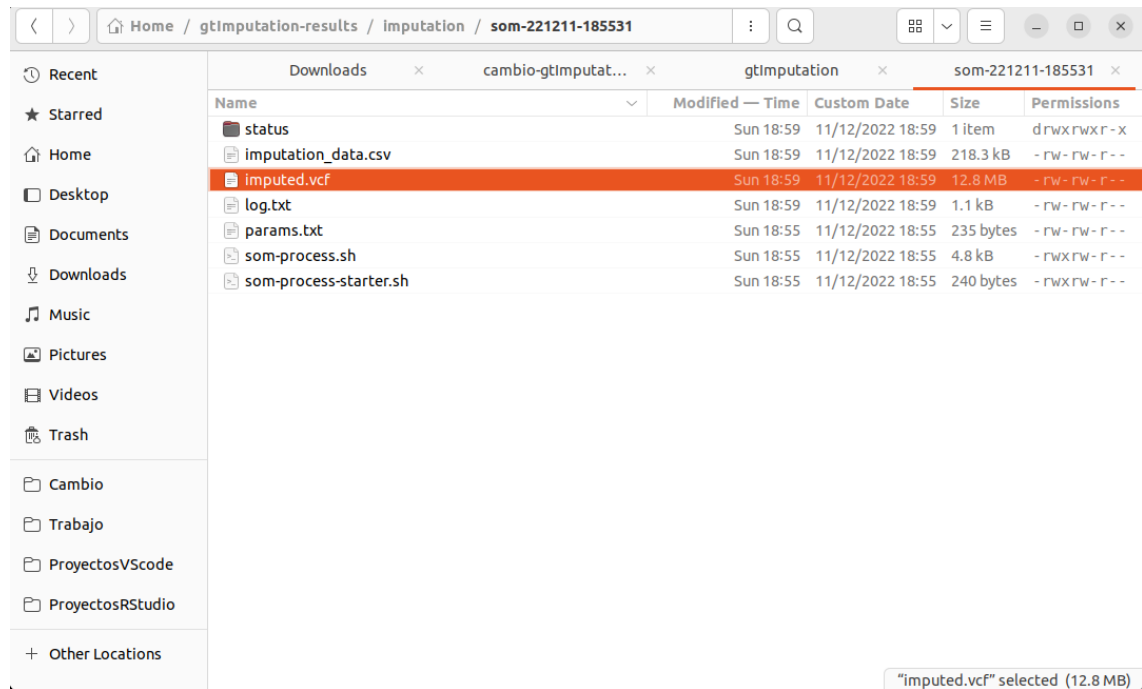*Main menu > Configuration > Browse gtImputation config file*

Figure. 26. Files generates in the result dataset directory **som-221211-185531**.

How to cite

If you are using gtImputation, you should cite the following paper:

Mora-Márquez F, Nuño J. C., Soto A & López de Hereda U (in review). Missing genotype imputation in non-model species using self-organizing maps.

References

Goudet, Kay, Weir - 2018 - How to estimate kinship - DOI: 10.1111/mec.14833

Ragsdale, Gravel- Unbiased Estimation of Linkage Disequilibrium from Unphased Data - DOI: 10.1093/molbev/msz265)

Vittigli, G. (2018). MiniSom: minimalistic and NumPy-based implementation of the Self Organizing Map. Available online at: https://github.com/JustGlowing/minisom/