README_ipyrad.md

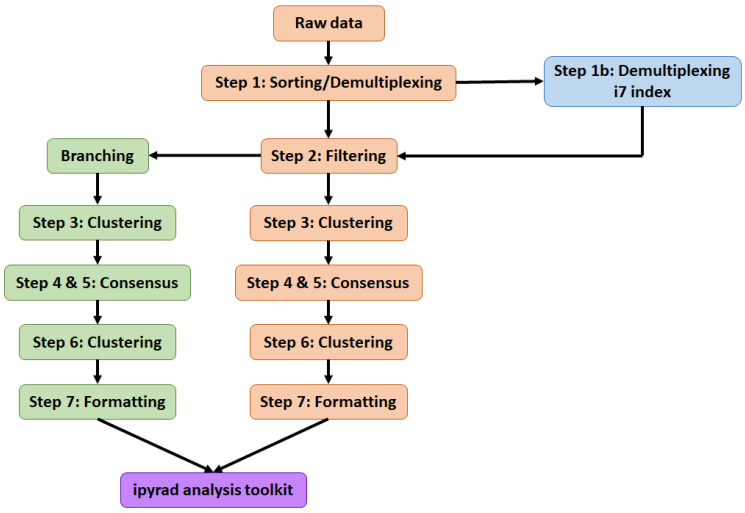
ipyrad_documentation

By Enora Geslain on 12/05/21

Reference: https://ipyrad.readthedocs.io and Eaton DAR & Overcast I. "ipyrad: Interactive assembly and analysis of RADseq datasets." Bioinformatics (2020).

You can find the script and data examples at this link: https://github.com/Enorya/LBEG_documentation/tree/main/ipyrad

Ipyrad is a pipeline which allows to call SNPs on different kind of data (rad, ddrad, gbs) but it also allows to do downstream analysis on the resulting SNPs calling (PCA, Structure analysis, phylogenetic analysis...)



ipyrad analysis

Preparing the working directory and the data

1. Create a new directory with a meaningful name (like "Trematomus_analysis_ipyrad") in order to put all your analysis with ipyrad in it.
2. Move in that new directory and create 4 folders: `scripts`, `raw`, `info` and a folder with a meaningful name in order to put your results (like "trematomus_results").
3. If you are working with paired-end data make sure the names of your raw files have the exact mentions `_R1_` and `_R2_` to differentiate forward and reverse reads (You will see examples in the folder `Example/raw/`). It is mandatory, otherwise the software is not going to differentiate the reads.
4. Depending on the barcoding technique you used you will need to create either 1 file or more files containing the barcodes. Look at the documentation in this location to find out how to create your barcodes file(s). When they will be created put them in the `info` folder. Take a look at the demultiplexing with i7 part of this document if you need to use this technique.

For the example dataset there is 8 different libraries and in each library the same set of double barcodes was used. Because of that the demultiplexing was done in 2 steps for each library. First, the reads were demultiplexed using the inline barcodes (8 different) with this barcodes file. Then, they were demultiplexed again using, the i7 index present in the line preceding the read's sequence (12 differents) with the files named `i7_index_inline?_lib1.txt` in this location

Connecting to the miniconda3 environment (to be able to use ipyrad)

1. Execute this command to connect to the environment: `source activate /staging/leuven/stg_00026/Softwares/miniconda3/`

Creating the parameter file (params file) and change it

1. Use the command `ipyrad -n meaningful_name` (this name will be used to name your outputs).
2. Open your new params file (its name will be "params-meaningful_name.txt", see the example) with your favourite text editor (nano for example).
3. Change the parameters at your convenience. Some parameters are mandatory, here you will see, for each step of the pipeline, the mandatory parameters marked with an asterisk.

At the beginning it can give the impression that there is too much parameters but don't panic! You can just let the default parameters for your first run and you will have the opportunity to do multiple other runs changing the parameters values with the branching.

Running the analysis

There is 2 possibilities:

1. Run the complete pipeline using the command `ipyrad -p params-meaningful_name.txt -s 1234567 -c 108 --MPI`
2. Run the pipeline step by step giving the number of the step you want to launch `ipyrad -p params-meaningful_name.txt -s 1 -c 108 --MPI`

You can notice the options `--MPI` and `-c 108` . The former indicates that you are going to parallelize your job (doing different steps at the same time). The latter gives the number of cores you will use for your job (it should correspond to the number of cores you have allocated for your job on the cluster). See the script example for a submission on the cluster.

I would recommend to do step 1 and 2, then do some statistics on your samples in order to exclude some if needed (using the branching method) and after that, go for the rest of the steps together.

Demultiplexing with i7

Open jupyter-notebook

1. Connect to the VSC using NoMachine
2. Open a terminal and begin an interactive session (you can change the walltime): `qsub -I -l nodes=1:ppn=10 -l walltime=3:00:00 -l pmem=10gb -A llp_lbeg`
3. Move to the directory you want to work in: `cd /path/to/your/folder`
4. Activate the conda environment in order to run ipyrad and jupyter-notebook: `conda activate /staging/leuven/stg_00026/Softwares/miniconda3`
5. Launch jupyter-notebook: `jupyter-notebook --ip name_of_the_core --port vsc_number`

(when looking at your terminal you will see something like `vsc34088@r22i27n08` where "name_of_the_core" corresponds to `r22i27n08` and "vsc_number" corresponds to `34088`)

6. Some text will appear, that's normal, don't panic! At the end you will see a message:

```
To access the notebook, open this file in a browser:
        file:///vsc-hard-mounts/leuven-user/340/vsc34088/.local/share/jupyter/runtime/nbserver-4368-open.html
```

Copy-paste the link in chrome browser, this will open the jupyter-notebook.

7. You will see the different folders or files of your working directory. Click on the button called `IPython Clusters` in the top left of your screen. In the column `# of engines` put the same number of cores you asked for your interactive session and click `Start` .
8. Move back to `Files` and create a new `Python 3 notebook` clicking on the button `New` in the top right of the screen.

You are now in a jupyter-notebook file!

Demultiplex your i7 index

1. In the first cell, import the different modules needed:

```
import ipyrad as ip
import ipyrad.analysis as ipa
import ipyparallel as ipp
import toyplot
```

Tip: To execute the cell rapidly you can press *Ctrl + Enter*

2. Add a new cell and verify on how many cores you are running:

```
ipyclient = ipp.Client()
print("Connected to {} cores".format(len(ipyclient)))
```

3. Add a new cell and do the different steps for the demultiplexing: set up the parameters (assembly name, project dir, path of the fastq files, path of the barcodes file, datatype), set hackers parameters to demultiplex on i7 index, run the demultiplexing.

```
# Set up your parameters
trem = ip.Assembly("trem_lib1_inl1_i7")
trem.params.project_dir = "/Example/trematomus_results/trem_lib1_inl1_i7"
trem.params.raw_fastq_path = "/Example/raw/trem_lib1_inl1_R*_.fastq.gz"
trem.params.barcodes_path = "/Example/info/i7_index_inline1_lib1.txt"
trem.params.datatype = "pairddrad"

# Set hackers params to demultiplex on i7 index
trem.hackersonly.demultiplex_on_i7_tags = True
trem.hackersonly.merge_technical_replicates = True

# Run the demultiplexing (step 1)
all_trem.run("1", ipyclient=ipyclient, force=True)
```

See the example (**Warning**: open it using jupyter-notebook otherwise it will not be readable)

Branching your analysis

Branching in order to redo some analysis changing some parameters

1. Use the command: `ipyrad -p params-meaningful_name.txt -b new_meaningful_name`
2. Change some parameters in your new params file.
3. Run the pipeline again from any step you want but add `--force` at the end of the command: `ipyrad -p params-new_meaningful_name.txt -s 567 -c 108 --MPI --force`

Branching in order to redo some analysis selectioning/excluding some individuals

There is 3 possibilities depending on the number of samples you want to keep:

1. To keep a very few number of samples: give the name of the samples you want to keep when creating the new params file (here 1A0, 1B0 and 1C0 are three different samples)

```
ipyrad -p params-meaningful_name.txt -b new_meaningful_name 1A0 1B0 1C0
```

2. To exclude a very few number of samples: give the name of the samples you want to exclude and add a dash when creating the new params file

```
ipyrad -p params-meaningful_name.txt -b new_meaningful_name - 1A0 1B0 1C0
```

3. To keep a very large number of samples: give a text file containing a list of the samples you want to keep (see the example) when creating the new params file

```
ipyrad -p params-meaningful_name.txt -b new_meaningful_name list_of_samples.txt
```

/!\ Warning: if you didn't change the parameter *project dir* the results of your new analysis will be in a new directory (named after your assembly name like `new_meaningful_name` here) inside your project directory (here `meaningful_name`).

ipyrad analysis toolkit

You can use the analysis toolkit of ipyrad with any kind of VCF files but in order to use them you first need to convert those files into hdf5 format. If you used ipyrad to do your analysis you will already have outputs in hdf5 format.

Converting VCF files into hdf5 files

1. Open a jupyter-notebook in your working directory following the steps here.
2. In the first cell, import the different modules needed:

```
import ipyrad.analysis as ipa
import pandas as pd
```

3. Add a new cell and compress your VCF file:

```
%%bash

# Load necessary modules
module purge
module load BCFtools/1.9-foss-2018a

cd /path/to/the/directory/containing/your/file/

# Compress the VCF file (creates .vcf.gz)
bgzip /path/to/your/file.vcf
```

4. Add a new cell and convert your file to hdf5:

```
# Initialize a conversion tool
converter = ipa.vcf_to_hdf5(
    name="meaningful_name",
    data="/path/to/your/file.vcf.gz",
    ld_block_size=20000,
)

# Run the converter
converter.run()
```

See the example (**Warning**: open it using jupyter-notebook otherwise it will not be readable)

PCA analysis

1. Open a jupyter-notebook in your working directory following the steps here. If your data are in VCF format you need to convert them into hdf5 using those explanations.
2. In the first cell, import the different modules needed:

```
import ipyrad.analysis as ipa
import pandas as pd
import toyplot
```

3. Add a new cell and setup the path to your vcf file converted to hdf5 format:

```
data = "/staging/leuven/stg_00026/enora/trem_ipyrad/new_trem/trem_default_popfile_outfiles/trem_default_popfile.snps.h
```

4. If you want to separate your samples into different populations, add a new cell and setup your populations:

```
imap = {
        "bor": ["P_bor_JRI_01", "P_bor_KGI_03"],
        "ber": ["T_ber_JRI_02", "T_ber_JRI_03", "T_ber_JRI_04", "T_ber_JRI_05", "T_ber_JRI_06_RS", "T_ber_JRI_06_b_RS"]
        "eul": ["Standart_2_lib1", "Standart_2_lib2", "Standart_2_lib3", "Standart_2_lib4", "Standart_2_lib5"],
        "tok": ["T_loe_PS96_369", "T_loe_PS96_370", "T_loe_PS96_371", "T_tok_PS96_173", "T_tok_PS96_345"],
}
```

You also have the opportunity to require a minimum percentage of samples that should have data in each group:

```
minmap = {i: 0.5 for i in imap}
```

5. Add a new cell and initiate the PCA analysis:

```
pca = ipa.pca(
    data=data,
    imap=imap,
    minmap=minmap,
    mincov=0.75,
    impute_method="sample",
)
```

mincov parameter is optional you can have more informations here. You can also change the impute_method, there is three methods described at this link.

6. Add a new cell and run the PCA:

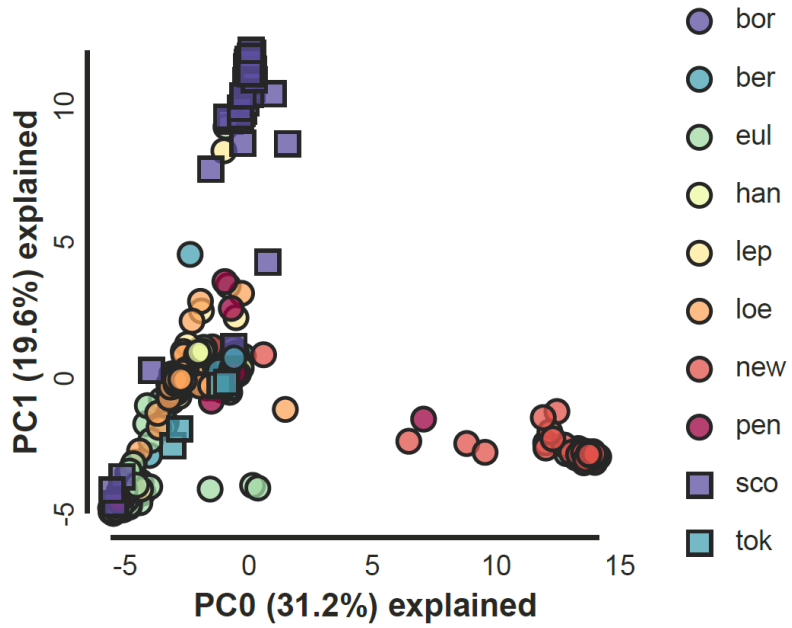
```
pca.run()
```

You have the possibility to write the PC axes to a CSV file following the instructions here.

7. Draw the resulting plot adding in the same cell:

```
pca.draw(0, 1);
```

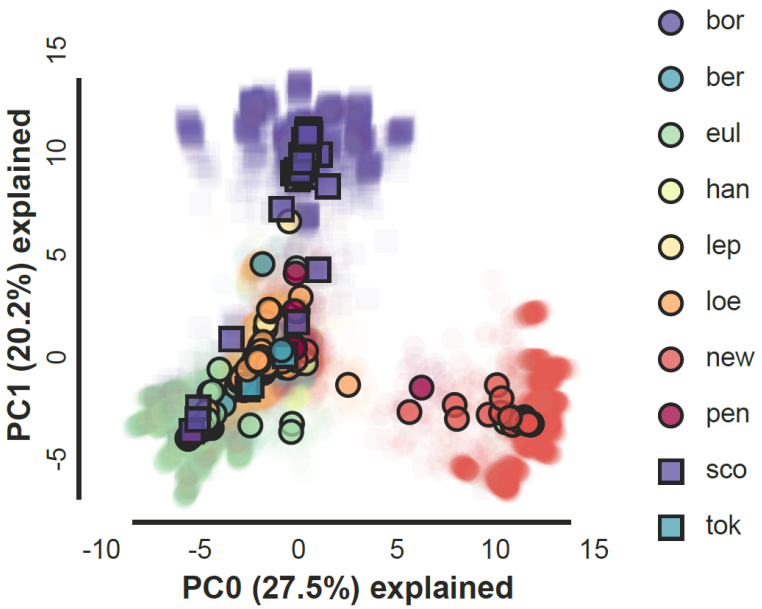
This command will plot the PCA results for the first 2 axes (numerotation of the axes begin at 0). You will end up with a plot looking like:



8. The PCA analysis of ipyrad toolkit take one random SNP on each locus for the plotting in order to have unlinked SNPs. You can replicate the analysis on different sets of random SNPs adding some parameters:

```
pca.run(nreplicates=25, seed=12345);
pca.draw(0, 1);
```

These commands will generate a plot with 25 replicates of the PCA (for the seed you can write any random number, it is just a marker for the analysis). With this command you will get a plot looking like the following, where the fuzzy points correspond to the replicates:



9. Finally, you can add a new cell to save your plot in PDF format:

```
pca.draw(0, 1, outfile="PCA-trem_default_axes0-1.pdf");
```

See the example (Warning: open it using jupyter-notebook otherwise it will not be readable)

Treemix analysis

1. Open a jupyter-notebook in your working directory following the steps here but instead of connecting to the *miniconda3* environment, connect to the *treemix* environment:

```
source activate /staging/leuven/stg_00026/Softwares/miniconda3/envs/treemix/
```

If your data are in VCF format you need to convert them into hdf5 using those explanations.

2. In the first cell, import the different modules needed:

```
import ipyrad.analysis as ipa
import toytree
import toyplot
import toyplot.pdf
```

3. Do step 3 and 4 as with the PCA analysis.
4. Add a new cell and initiate the Treemix analysis:

```
tmx = ipa.treemix(
    data=data,
    imap=imap,
    minmap=minmap,
    seed=123456,
    root="bor",
    m=2,
)
```

The `m` parameter corresponds to the number of migration events you are allowing for your analysis.

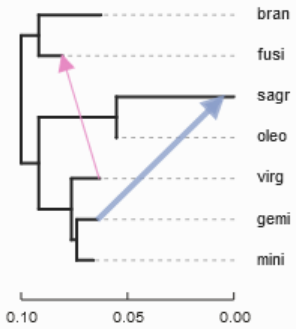
5. Add a new cell and run the Treemix analysis:

```
tmx.run()
```

6. Draw the resulting tree:

```
tmx.draw_tree()
```

You will get a plot like the following:



7. If you want you can add a new cell and draw a covariance matrix:

```
tmx.draw_cov()
```

8. In order to find the best value for the number of migration events `m` add a new cell and initiate a new treemix analysis:

```
tmx = ipa.treemix(
    data=data,
    imap=imap,
    minmap=minmap,
    seed=1234,
    root="bor",
)
```

Then add a new cell and iterate your run with different values of `m` :

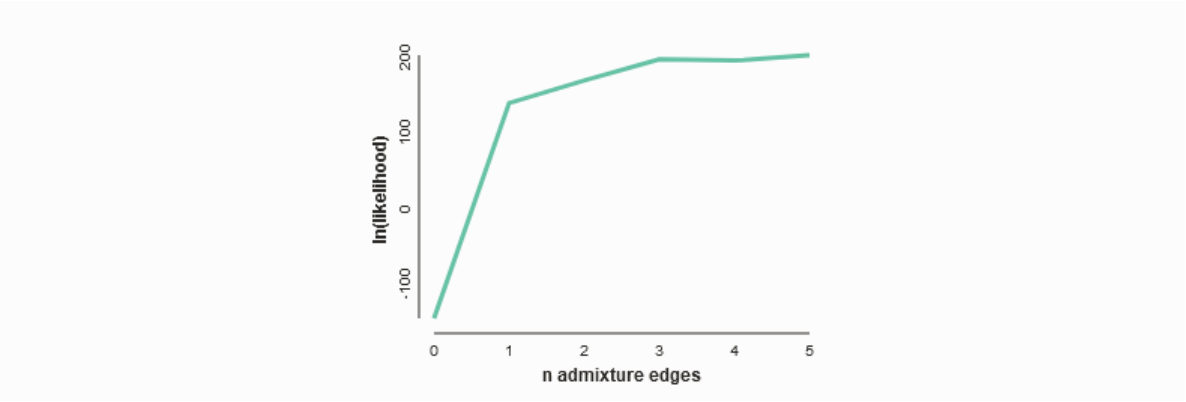
```
tests = {}
nadmix = [0, 1, 2, 3, 4, 5, 6]

for adm in nadmix:
    tmx.params.m = adm
    tmx.run()
    tests[adm] = tmx.results.llik
```

Finally, add a new cell again and plot the likelihood for these different values of `m` :

```
toyplot.plot(
    nadmix,
    [tests[i] for i in nadmix],
    width=350,
    height=275,
    stroke_width=3,
    xlabel="n admixture edges",
    ylabel="ln(likelihood)",
);
```

You will get a plot like the following one. All you have to do is choose the value at which the curve starts to flatten out.



9. Add a new cell and run again your analysis with the correct value but, this time, replicating on different set of random SNPs:

```
# Initiate a gridded canvas to plot trees on
canvas = toyplot.Canvas(width=600, height=700)

# Iterate on different set of random SNPs (9 here)
for i in range(9):

    # Initiate a treemix analysis object with a random seed
    tmx = ipa.treemix(
        data=data,
        imap=imap,
        minmap=minmap,
        root="bor",
        global_=True,
        m=3,
        quiet=True
    )

    # Run model fit
    tmx.run()

    # Select a plot grid axis and add tree to axes
    axes = canvas.cartesian(grid=(3, 3, i))
    tmx.draw_tree(axes)
```

10. Add a new cell and save your plot in PDF format:

```
toyplot.pdf.render(canvas, "treemix-m3.pdf")
```

See the example (Warning: open it using jupyter-notebook otherwise it will not be readable)