Development of an API for miRNA sequencing data that converts mirGFF3 files to VCF

Roderic Espín Garcia
Máster en Bioinformática y Bioestadística
Área 1 - Subárea 9: Análisis de secuencias de RNA reguladores

Lorena Pantano Rubiño
Maria Jesús Marco Galindo

January 2nd, 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Development of an API for miRNA sequencing data that converts mirGFF3 files to VCF*
Nombre del autor:	*Roderic Espín Garcia*
Nombre del consultor/a:	*Lorena Pantano Rubiño*
Nombre del PRA:	*Maria Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	*Máster en Bioinformática y Bioestadística*
Área del Trabajo Final:	*Análisis de secuencias de RNA reguladores*
Idioma del trabajo:	*Inglés*
Palabras clave	*VCF, miRNA, tool*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Los microARN (también llamados miARN) son pequeñas moléculas de ARN de unos 20-25 nucleótidos de longitud y que están envueltos en la regulación de genes postrancripcionales. Los miARN son esenciales en casi cualquier proceso biológico y se le ha asociado con varias enfermedades humanas.

Las secuencias con variaciones respecto al miARN de referencia se llaman isomiR. Los isomiRs también pueden depender del género y la raza.

Actualmente, existe un proyecto llamado mirtop que unifica la investigación relativa de los miARN e isomiRs cuyo objetivo es optimizar los análisis de miARN y promover el desarrollo de herramientas analíticas de tipo downstream. En este proyecto se ha originado un nuevo formato llamado mirGFF3, para la salida de resultados de detección y cuantificación de miARN/isomiR.

El objetivo de este trabajo es añadir a mirtop una herramienta que convierta ficheros mirGFF3 al formato VCF (Variant Call Format). En este formato se almacenan las variantes genéticas respecto al genoma de referencia y es muy utilizado como entrada en otras herramientas bioinformáticas.

Se ha utilizado Python como lenguaje de programación para la realización de la herramienta y se utilizarán datos de muestras sintéticas y reales (ya en formato mirGFF3) para comprobar su funcionamiento y posterior análisis mediante gráficos, también generados por Python, aunque este análisis es independiente del proyecto mirtop.

Abstract (in English, 250 words or less):

microARN (also called miARN) are short RNA molecules with a nucleotides length of 20-25 that regulates gene expressions. miRNAs are essential to all biological processes and its associated with several human diseases.
isomiRs are sequence variants from annotated miRNAs. isomiRs can also depend on sex or ethnicity.

Currently, a project called mirtop unites research of miRNAs and isomiRs with the aim of promoting the development of downstream analysis tools. In that project a new format called mirGFF3 was originated, for the output of miRNA/isomiR detection and quantification results.

The main goal of this project is to add a tool to the mirtop API that converts mirGFF3 format files to VCF (Variant Call Format). In this format, genetic variants respect the reference genome are stored, and is a file format very used in other useful bioinformatics tools.

Python was the programming language chosen to implement the conversion tool. Synthetic and real samples will be used (already in mirGFF3 format) to test the functionality. A subsequent analysis (also with python) will be carried out by plotting graphs, although this analysis is independent from the mirtop project.

Table of contents

Summary of figures

Summary of tables

1. Introduction

1.1 Context and justification of the project

1.1.1 General description

With the emergence of Next Generation Sequencing (NGS) technologies, RNA transcriptomes were massively generated. This project specifically focuses on miRNA (microRNA), which are short RNA molecules with a nucleotides length of 20-25 that regulates gene expressions and its identification is important to understand gene regulations. [1] [2]

In that context, an international collaboration implemented an API (miRTop) developed in python where one of its functions converts several miRNA quantification outputs to the mirGFF3 format [3] [4] .The approach of this project is to further develop the current pipeline and add a new tool that converts this same mirGFF3 format to a Variant Call Format (VCF) [5], so it can be imported from other tools which specializes in genomic variation analysis.

A relevant section of this project is also to execute the tool with small non-coding RNA synthetic samples [6] to measure the accuracy of single nucleotide changes (SNVs) by the sequencing technology. This will allow to estimate the false discovery rate that can be used as a cutoff to filter out SNVs or biological samples.

1.1.1.1 miRNA and isomiRs

microRNAs (miRNAs) are small RNA molecules of 20-25 nucleotides long. miRNA genes are transcribed into a primary RNA (pri-miRNA) that is processed into a hairpin miRNA precursor after cutting off the 5’ and 3’ tails by Drosha and DGCR8 proteins [7]. Then, the hairpin is exported to the cytoplasm and processed by Dicer, which cleaves off the hairpin loop and releases a miRNA duplex about 22 nucleotides long [8]. miRNAs are essential to all biological processes, like cell differentiation, cell proliferation and cell death [9] [10]. The deficit of excess of miRNAs has effects with several human diseases and different types of cancer [11] .

isomiRs are sequence variants from annotated miRNAs [12]. They were described first by Morin et al in human stem cells using NGS technologies [2]. As consequence of Biochemical processes, sequence variations can affect the mature miRNA sequence [13]. The imprecision of the Drosa/Dicer cutting could make variations at 3’ and 5’ ends, but these variations constitutive in both pathients and healthy individuals [14] [15]. isomiRs depend of the subject, that is, its sex or ethnicity [14], but also disease subtype [16] [17], although specific functions of isomiRs is currently not known well [18].

1.1.2 Justification of the Master's Final Project

The main purpose of this project is the creation of a tool that will be able to facilitate the research of miRNAs and contribute to their community. With the implementation of the mirtop API and its mirGFF3 format for miRNAs, it led to the possibility to continue its pipeline and create a tool which converts this format to a variant generic one (VCF). This way, the last file of the pipeline can be used in other tools specialized in genomic variation.

1.2 Objectives

1.2.1 General objectives

1) Build and incorporate to the miRTop pipeline a python tool that converts mirGFF3 miRNA files to VCF.

2) Test the conversion and do an analysis using a published dataset that contains real samples and synthetic samples.

3) If the existence of SNVs is confirmed, determine the false positives rate and apply a methodology to filter them out.

1.2.2. Specific objectives

- Build and incorporate to the miRTop pipeline a python tool that converts GFF3 miRNA files to VCF.

  1) Determine the attributes, characteristics of both mirGFF3 and VCF formats and the relationships between them.

  2) Build a python tool that converts mirGFF3 format to VCF in a standalone style.

  3) Include the tool to the miRTop project.

  4) Build a unit test to check the code does what is expected.

  5) Coordinate the versions and improvements of the conversion tool to the project's GitHub repository.

- Test the conversion and do an analysis using a published dataset that contains real samples and synthetic samples.

  6) Analyze public data set to get mirGFF3 files.

7) Convert to VCF

8) Check files integrity with a validator tool.

9) Fix potential bugs that appear during the process.

- If the existence of SNVs is confirmed, determine the false positives rate and apply a methodology to filter them out.

10) Plot SNV supporting reads for each negative and read samples

11) Compare SNV rate to sequencing error rate

12) Find a characteristic that could filter out majority of SNV from synthetic control samples and apply to real samples.

1.3 Approach and method to follow

This project is clearly centralized in the tool that converts mirGFF3 files to a VCF format. The VCF format has been chosen since it's the standard file in communities to annotate the genetic variations. The mirGFF3 has been selected as the input format because it is the output format of the current pipeline of the miRTop project, since its format content seeks to standardize miRNA.

The next step is to select the programming language in order to develop the tool. Although at the beginning it will be a standalone (independent and without the need of an internet connection), afterwards the code will be slightly modified and added to the miRTop project. Because the current project is being developed in Python, this will be the language being used.

Before developing the code of the tool, it is important to determine the attributes of the mirGFF3 and VCF formats and the relationships between them, in order to adapt them correctly.

Once the tool is completed, it will be executed first using synthetic and real samples and a VCF format file will be generated and may contain SNVs. If there isn’t, it would mean there are no variations, although this is practically impossible because sequencing error is expected. [7]

Assuming that SNVs are to be found, false positives must be determined and, finally, if their existence is evidenced, a methodology will be applied to try to prevent them or, at least, reduce its rate of appearance. But this part may be out of scope of this project.

1.4 Planning

The project is divided into tasks, which are based on the specific objectives that have been previously described. A schedule has been determined based on the tasks which knowledge is consolidated and the ones that need more research in order to achieve the best results. In that sense, the tasks and its schedule are as follows:

1.4.1 Tasks

Task 1: Establish the relationships and formatting between mirGFF3 and VCF.

Task 2: Build an independent tool developed in Python that converts mirGFF3 files to VCF.

Task 3: Unit testing the tool to assure proper functionality.

Task 4: Adapt the tool to the miRTop project so it will be able to continue its pipeline.

Task 5: Coordinate everything in the GitHub's repository.

Task 6: Test the tool with the published dataset.

Task 7: SNV calling the variations of the VCF file to determine whether there are false positives.

Task 8: Analyze the false SNVs in order to determine (if able) its origin.

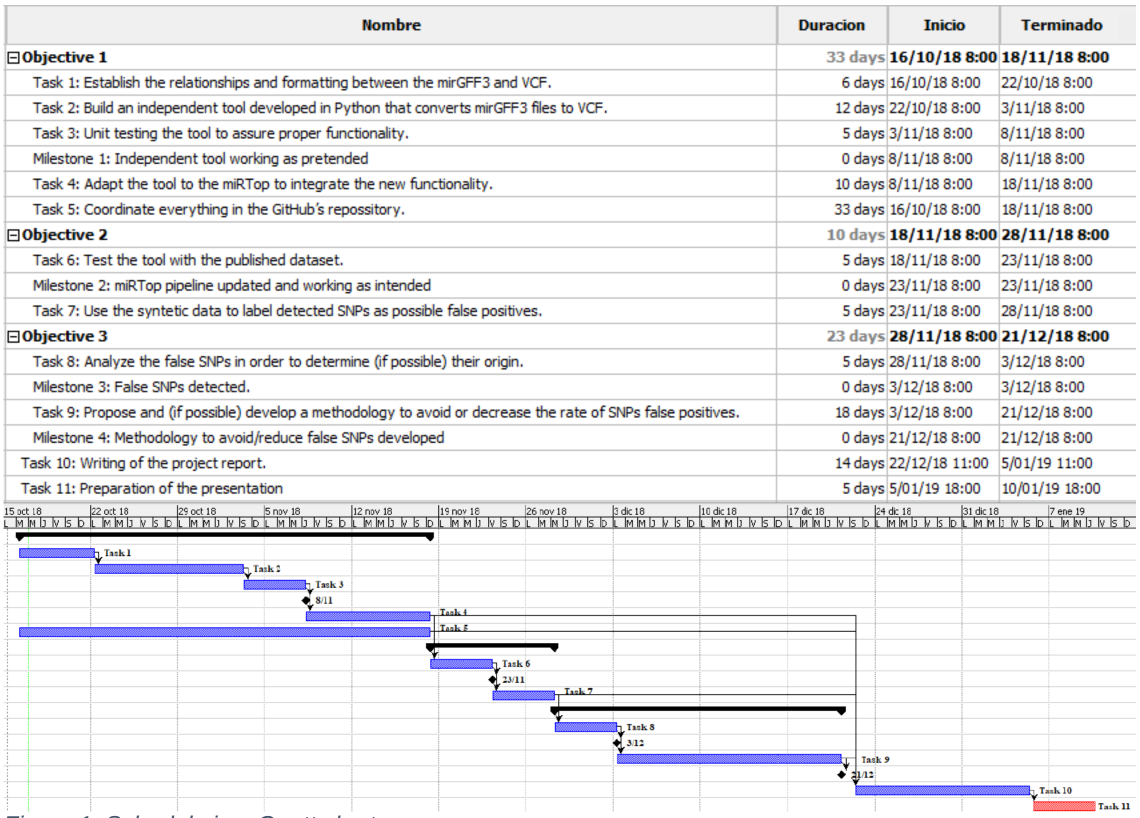
Task 9: Propose and (if possible) develop a methodology to avoid or decrease the rate of false positives SNVs.

Task 10: Writing of the project report.

Task 11: Preparation of the presentation.

1.4.2. Schedule

The schedule of this project has been defined with a Gantt chart and is as follows:



Nombre	Duracion	Inicio	Terminado
Objective 1	33 days	**16/10/18 8:00**	**18/11/18 8:00**
Task 1: Establish the relationships and formatting between the mirGFF3 and VCF.	6 days	16/10/18 8:00	22/10/18 8:00
Task 2: Build an independent tool developed in Python that converts mirGFF3 files to VCF.	12 days	22/10/18 8:00	3/11/18 8:00
Task 3: Unit testing the tool to assure proper functionality.	5 days	3/11/18 8:00	8/11/18 8:00
Milestone 1: Independent tool working as pretended	0 days	8/11/18 8:00	8/11/18 8:00
Task 4: Adapt the tool to the miRTop to integrate the new functionality.	10 days	8/11/18 8:00	18/11/18 8:00
Task 5: Coordinate everything in the GitHub's repossitory.	33 days	16/10/18 8:00	18/11/18 8:00
Objective 2	10 days	**18/11/18 8:00**	**28/11/18 8:00**
Task 6: Test the tool with the published dataset.	5 days	18/11/18 8:00	23/11/18 8:00
Milestone 2: miRTop pipeline updated and working as intended	0 days	23/11/18 8:00	23/11/18 8:00
Task 7: Use the syntetic data to label detected SNPs as possible false positives.	5 days	23/11/18 8:00	28/11/18 8:00
Objective 3	23 days	**28/11/18 8:00**	**21/12/18 8:00**
Task 8: Analyze the false SNPs in order to determine (if possible) their origin.	5 days	28/11/18 8:00	3/12/18 8:00
Milestone 3: False SNPs detected.	0 days	3/12/18 8:00	3/12/18 8:00
Task 9: Propose and (if possible) develop a methodology to avoid or decrease the rate of SNPs false positives.	18 days	3/12/18 8:00	21/12/18 8:00
Milestone 4: Methodology to avoid/reduce false SNPs developed	0 days	21/12/18 8:00	21/12/18 8:00
Task 10: Writing of the project report.	14 days	22/12/18 11:00	5/01/19 11:00
Task 11: Preparation of the presentation	5 days	5/01/19 18:00	10/01/19 18:00

Figure 1: Schedule in a Gantt chart

Total: 85 days.

1.4.3 Milestones

Milestone 1: Independent tool working as pretended
Milestone 2: miRTop pipeline updated and working as intended
Milestone 3: False SNVs detected.
Milestone 4: Methodology to avoid/reduce false SNVs developed

1.4.4 Risk analysis

While there may always be some kind of risk in the tasks relative to the first general objective of the project, such as difficulties involving conversion between formats or the adaptation from standalone to the miRTop pipeline, the riskiest tasks are part of the second and third objectives.

In particular, when determining false SNVs, a risk can be found in the fact that, with the converted samples in the VCF format, the false positives may not be detected (if they are).

Another risk can happen when establishing a methodology that removes/reduces false SNVs (and not the “True” ones), because without analyzing the data in advance, it is difficult to determine to what extent it is possible to achieve it.

The second risk may be solved by applying existing methodologies that actually diminishes false SNVs (although it will involve further research).

1.5 Brief summary of the results

The results obtained from this project are the python scripts that converts mirGFF3 format data to VCF, and another that reads VCF files data and generates graphics to interpret the SNVs found in the mirGFF3 file. Two VCF files are also created from synthetic and real samples.

1.6 Brief description of other chapters

The main chapters of this project are:

- *mirGFF3 and VCF formats*. In this chapter the format of the mirGFF3 and VCF files are explained and how they relate.

- *Methods and results*. All the functions of the script that does the conversion are explained in this chapter, it also includes some plots to interpret the data results of the samples. Finally, there is a section that indicates all the produced results of the project.

2. mirGFF3 and VCF formats

2.1 The mirGFF3 format

mirGFF3 [3] is a text file format and is based on the GFF3 format and adapted and focused to miRNA data. Its purpose is to store positional features [8] and is widely used by many software [9] and it can represent many situations like non-coding transcripts or alignments. The main idea of the mirGFF3 format (which of course will contain miRNA data) is to promote sharing, re-analysis and developing of downstream analysis.

All information found inside a mirGFF3 format file is divided into two sections: headers, which are the meta-information of the data, whose lines always start with ##. After the meta-information, every line will show a sequence that maps a miRNA.

2.1.1 Headers

The following table shows the headers, its description, if it is required and an example:

Headers of mirGFF3			
Name	**Description**	**Required**	**Example**
VERSION	The version of mirGFF3	Yes	## mirGFF3. VERSION 1.1
source-ontology	The database source ontology	Yes	## source-ontology: miRBasev21 doi:10.25504/fairsharing.hmgte8
CMD	Commands used to generate the file	No	
REFERENCE	Genome/Database version used	No	
COLDATA	Samples names used. Separated by a comma.	Yes	## COLDATA TruSeq_Lab1_SynthEQ-mirbase-ready, TruSeq_Lab2_SynthEQ-mirbase-ready
FILTER	Meaning of the “FILTER” tags	No	## FILTER: PASS(ok), REJECT(no ok)

Table 1: Headers of mirGFF3, adapted from miRTop - mirGFF3 format [3]

2.1.2 Columns and its data

The fields that are tab-separated and contains information and the fields that do not have any must be informed with a dot. The description of the columns is as follows:

Columns of mirGFF3				
Name	**Column number**	**Description**	**Required**	**Example**
seqID	1	Precursor name	Yes	hsa-mir-675
source	2	Database used for the annotation with its version	Yes	miRBasev21
type	3	Type of miRNA (reference miRNA, isomiR, etc)	Yes	isomiR
start	4	Precursor start position.	Yes	10
end	5	Precursor end position.	Yes	31
score	6	Mapping score or other type of score	No	0
strand	7	If the strand is sense (+) or antisense (-)	Yes	+
phase	8	It is currently ignored.	No	.
attributes	9	Contains the relevant attributes of the specific read	Yes	Read=TGTGCGGAGAGGGCCCACAGGG; UID=iso-22-9V62R12P4; Name=hsa-miR-675-5p; Parent=hsa-mir-675; Variant=iso_5p:+1,iso_snv; Cigar=T19MGM; Expression=2,0,0,2;

Table 2: Columns of mirGFF3, adapted from miRTop - mirGFF3 format [3]

The ninth column attributes, which contains a specific set of feateures separated by semicolon. The order of the fields is not important and their information is:

Attributes of mirGFF3			
Name	**Description**	**Required**	**Example**
UID	Unique ID based on its sequence.	Yes	iso-22-66P5J2RIM
Read	Read sequence	No	TGTGCGGAGAGGGCCCACAGGG
Name	Mature name	Yes	hsa-miR-675-5p
Parent	Hairpin precursor name	Yes	hsa-mir-675
Variant	Categorical types, adapted from isomiR-	Yes	iso_5p:+1,iso_snv

	SEA [10]. There are 8 variant categories. Explained with more detail in Table X		
Changes	Indicates nucleotides being changed	No	Changes=iso_5p=t, iso_snv:g
Cigar	CIGAR string [11]	Yes	T19MGM
Hits	Number of hits in the database	Yes	1
Alias	Names from the database, separated by a comma	No	MIMAT0004284
Genomic	Positions on the genome. Format: chr:start-end,chr:start-end	No	11:1996771-1996792(+)
Expression	Raw counts separated by a comma	Yes	2,0,0,2;
Filter	Either Pass or Reject, used to filter the sequence	Yes	Pass
Seed_fam	Currently not being used	No	

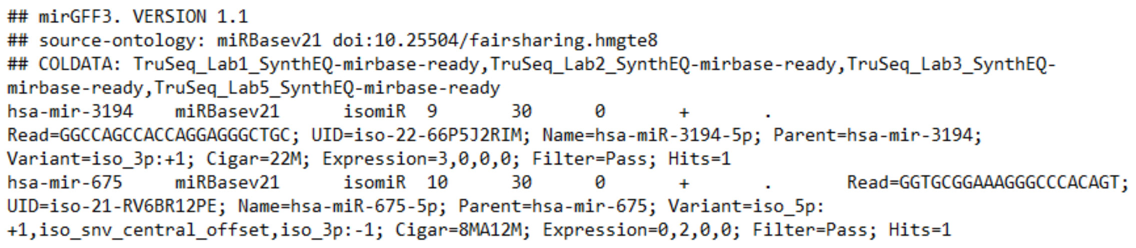
Table 3: Attributes columns description of mirGFF3, adapted from miRTop - mirGFF3 format [3]

In the attributes column there is a field called “variants” that show affected nucleotides in the read sequence, their definition is:

Categories of ‘Variants’	
Name	**Description**
iso_5p	Extra nucleotides not included in the reference miRNA, negative or positive values depends on whether the sequence starts n nucleotides after (-) or before(+) the reference.
iso_3p	Same explanation as above.
iso_add_3p	Same explanation as above.
iso_snv_seed	The affected nucleotides are between the positions 2 and 7.
iso_snv_central_offset	The affected nucleotide is at the position 8.
iso_snv_central	The affected nucleotides are between the positions 9 and 12
iso_snv_central_supp	The affected nucleotides are between the positions 13 and 17.
iso_snv	Anything else

Table 4: Variants categories, adapted from miRTop - mirGFF3 format

An example of a few mirGFF3 lines, as well as the meta-information:



```
## mirGFF3. VERSION 1.1
## source-ontology: miRBasev21 doi:10.25504/fairsharing.hmgte8
## COLDATA: TruSeq_Lab1_SynthEQ-mirbase-ready,TruSeq_Lab2_SynthEQ-mirbase-ready,TruSeq_Lab3_SynthEQ-
mirbase-ready,TruSeq_Lab5_SynthEQ-mirbase-ready
hsa-mir-3194    miRBasev21      isomiR  9       30      0       +       .
Read=GGCCAGCCACCAGGAGGGCTGC; UID=iso-22-66P5J2RIM; Name=hsa-miR-3194-5p; Parent=hsa-mir-3194;
Variant=iso_3p:+1; Cigar=22M; Expression=3,0,0,0; Filter=Pass; Hits=1
hsa-mir-675     miRBasev21      isomiR  10      30      0       +       .       Read=GGTGCGGAAAGGGCCCACAGT;
UID=iso-21-RV6BR12PE; Name=hsa-miR-675-5p; Parent=hsa-mir-675; Variant=iso_5p:
+1,iso_snv_central_offset,iso_3p:-1; Cigar=8MA12M; Expression=0,2,0,0; Filter=Pass; Hits=1
```

Figure 2: mirGFF3 example

2.2 The VCF

VCF [12] is a text file format that contains variation in the sequence and its information. It uses meta-information, a header line and the variants with the supporting information. The current version and the one used in this project is the 4.3.

2.2.1 Meta-information lines

These lines are included after a double hash sign (##) and are optional except the file format which is always required. Some of these are as follows:

Meta-information of VCF		
Name	**Description**	**Format/example**
fileformat	Details the version number of the VCF	##fileformat=VCFv4.3
fileDate	Date of file creation	##fileDate=20181210
Source	The database source	##source=miRBasev21 doi:10.25504/fairsharing.hmgte8
INFO	Information fields that are included in the data	##INFO=<ID=ID,Number=number,Type=type",Description="description",Source="source",Version="version">
FILTER	Filters applied to the data	##FILTER=<ID=ID,Descroption=description>
FORMAT	Genotype fields specified	##FORMAT=<ID=ID,Number=number,Type=type,Description="description">

Table 5: Meta-informaton of VCF, adapted from the VCF specifications

2.2.2 Header line

It's the header line and it contains the name of the 8 required columns plus the optional ones. The mandatory and tab-separated fields are: CHROM, POS, ID, REF, ALT, QUAL, FILTER and INFO and it must start with a hash sign (#)

2.2.3 Data lines

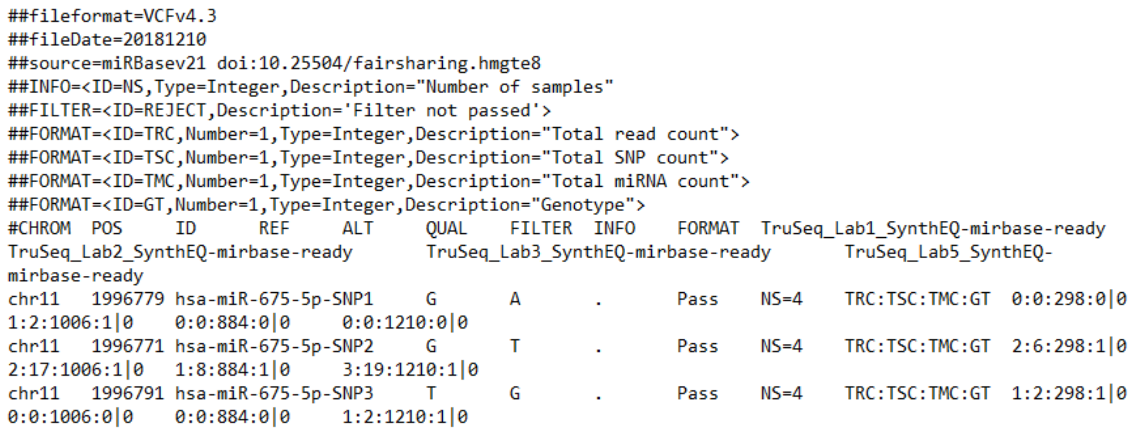
They are tab-limited and missing values are stated with a dot (**.**).

Data lines of VCF		
Name	**Description**	**Example**
CHROM	Indicates the chromosome of reference	11
POS	The reference position, the first position being 1	1996779
ID	Identifier, unique per data record (no duplicates allowed)	hsa-miR-675-5p-SNP1
REF	Reference base/s	G
ALT	Alternate base/s	A
QUAL	Quality, in a phred-scaled score.	7
FILTER	Filter status	Pass
INFO	Additional information, its description must be informed in the meta-information lines	NS=4
FORMAT	Not required, they are the genotype fields indicated in the meta-information lines	TRC:TSC:TMC:GT

Table 6: Data lines of VCF, adapted from the VCF specifications

For each sample another column can be added with information relative to the FORMAT genotype fields.

An example of a few VCF lines, as well as its meta-information and header:



```
##fileformat=VCFv4.3
##fileDate=20181210
##source=miRBasev21 doi:10.25504/fairsharing.hmgte8
##INFO=<ID=NS,Type=Integer,Description="Number of samples"
##FILTER=<ID=REJECT,Description='Filter not passed'>
##FORMAT=<ID=TRC,Number=1,Type=Integer,Description="Total read count">
##FORMAT=<ID=TSC,Number=1,Type=Integer,Description="Total SNP count">
##FORMAT=<ID=TMC,Number=1,Type=Integer,Description="Total miRNA count">
##FORMAT=<ID=GT,Number=1,Type=Integer,Description="Genotype">
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  TruSeq_Lab1_SynthEQ-mirbase-ready
TruSeq_Lab2_SynthEQ-mirbase-ready       TruSeq_Lab3_SynthEQ-mirbase-ready       TruSeq_Lab5_SynthEQ-
mirbase-ready
chr11   1996779 hsa-miR-675-5p-SNP1     G       A       .       Pass    NS=4    TRC:TSC:TMC:GT  0:0:298:0|0
1:2:1006:1|0    0:0:884:0|0     0:0:1210:0|0
chr11   1996771 hsa-miR-675-5p-SNP2     G       T       .       Pass    NS=4    TRC:TSC:TMC:GT  2:6:298:1|0
2:17:1006:1|0   1:8:884:1|0     3:19:1210:1|0
chr11   1996791 hsa-miR-675-5p-SNP3     T       G       .       Pass    NS=4    TRC:TSC:TMC:GT  1:2:298:1|0
0:0:1006:0|0    0:0:884:0|0     1:2:1210:1|0
```

Figure 3: VCF example

2.3 Adaptation from the mirGFF3 to the VCF Format

Now that the fields of both formats are known, the next step is to link the corresponding fields between them and adapt, if necessary, the format of the original data to the new one.

The meta-information lines of the VCF format does not need the mirGFF3 format to complete it, the only exception being the ##source line, and is the same value as the source in the mirGFF3 format.

Data lines are not directly related to the ones of the VCF format, so it is necessary to establish their relationship as follows:

Relationship between VCF data lines and mirGFF3		
VCF field	**mirGFF3 fields involved**	**Details**
CHROM	Parent (hairpin precursor name)	From the precursor name, there is a function in the miRTop project that resturns the chromosome and its relative position from the reference for this precursor.
POS	seqID, Read, Name, Parent, Variant, Cigar and start	From all these fields, it is determined the reference position, its nucleotides base/s and its alternate/s. A more deep explanation is found in the code functions.
ID	Parent (hairpin precursor name)	It uses the precursor name and the number of variations found to that moment.
REF	seqID, Read, Name, Parent, Variant, Cigar and start	Same as “POS” field
ALT	seqID, Read, Name, Parent, Variant, Cigar and start	Same as “POS” field
QUAL	N/A	Not used, no data to extract this information.
FILTER	Filter	Same information
INFO	COLDATA	In COLDATA there is the number and name of the samples involved. In INFO there will be the information relative to the number of samples.
FORMAT	Expression	The FORMAT field will have four types of information, TRC (Total read counts), TSC (Total SNV counts), TMC (Total miRNA counts) and GT (Genotype information). They are all based of the expression values, but indirectly uses other fields used to detect the SNVs

3. Methods and results

In this chapter, the functions involved in the generation of the VCF format will be explained.

3.1 Explanation of the script functions

Function	**expand_seqs**
Input	cigar (string)
Output	cigar_exp_read (string), cigar_exp_ref (string)
Description	Extends the CIGAR string and adapts it for the read and reference sequence

Table 7: Expand_seqs function

Because the CIGAR string [1] is the sequence associated to the alignment that has been applied previously in the current mirGFF3 format, there is a necessity in our code to adapt that CIGAR string to guess the reference sequence length used. The operations of the CIGAR strings that will be treated are “aligned match” (M), “insertion to the reference” (I), “deletion from the reference” (D), and a “sequence mismatch”, that is T, C, G or A.

In this same document [1] there is a table (where the description and symbols of CIGAR string can be found) with a column that indicates if the operation causes the alignment to step along, this is very important because this function will divide the CIGAR into two expanded CIGARs to match the read and reference sequences. Deletions, matches and mismatches won’t affect the reference position, but Insertions will do so, at the time of separating the string into two, insertions will go one way (reference) and deletions to the other (read). For better visualization, an easy example is explained below:

-From the input CIGAR string (e.g. 17MIII2MDD) the function will first extend it reading character by character, where only matches (“M”) will be preceded by digits (which indicates matches in succession), with the exception of “1”. Finally, the result of the extension is “MMMMMMMMMMMMMMMMMIIIMMDD”. To separate it, as it has been previously indicated, deletions will go to reference side and insertions to read ones. That is:

cigar_exp_read = MMMMMMMMMMMMMMMMMIIIMM
cigar_exp_ref = MMMMMMMMMMMMMMMMMMMMDD

This function (*expand_seqs*) is used before the *adapt_refseq*:

Function	**adapt_refseq**
Input	cigar_ref (string), hairpin (string), parent_ini_pos (integer), var5p (integer)
Output	refseq (string),

Description	Generates the reference sequence from where the read will be aligned

Table 8: adapt_refseq function

cigar_ref variable is the output of the function *expand_seqs* and is an extended CIGAR string used to interpret the length of the reference sequence.

The hairpin string, originally a file which contained all the sequences of miRNA hairpins in a FASTA format, is the hairpin of the current miRNA (the one being processed).

Parent_ini_pos is the integer value of the start of the reference sequence of the parent (relative to the hairpin).
var5p it is a variant originated from *iso_5p*, indicates the number of nucleotides differing to the reference's 5' position (in the read sequence).

This function finds the starting position of the reference sequence from the variables above mentioned. Hence, the index of the hairpin string is *parent_ini_pos* + *var5p* (can have a negative value), the end position will be indicated by the cigar extended reference.

A case illustrating the mentioned task, using the same CIGAR string of the *expand_seqs* function example:

*cigar_ref = MMMMMMMMMMMMMMMMMMMDD
*var5p = 1
*parent_ini_pos = 40
*hairpin = TATCAATAAGCCTTCTCTTCCCAGTTCTTCTTGGAGTCAGG AAAAGCTGGGTTGAGAGGAGCAGAAAAGAAANNNNNNNNNNN

The starting position will be 40+1=41 (marked in red). The length of cigar_ref is 21, so 21+41 = 62 is the end position of the reference sequence (marked in blue). Finally, the reference sequence of this lecture is AAAAGCTGGGTTGAGAGGAGC.

Again, this function is another precursor of the main function which calculates the SNVs and its positions, that is the cigar_2_key function.

Function	**cigar_length**
Input	cigar (string)
Output	total_n
Description	Returns the CIGAR length based on nucleotides

Table 9: cigar_length function

This function was only used for testing. There was a necessity to validate that all the read sequences of the lectures in the mirGFF3 file had the same length than the CIGAR. That is because the logic of the script is based entirely on the CIGAR string and the read sequence.

Function	cigar_2_key
Input	cigar_read (string), cigar_ref (string), readseq (string), refseq (string), pos (integer)
Output	Key_pos (list of strings), key_var (list of strings), ref (list of strings), alt (list of strings)
Description	Generates a list which contains all the SNVs found in the current lecture (key_var), the positions of these SNVs (key_pos) and their reference (ref) and alternate (alt) bases

Table 10: cigar_2_key function

The inputs needed for this function are: *cigar_read* and *cigar_ref* are the outputs of the *expand_seqs* function (they are the expanded CIGAR strings), the *readseq* string is the read sequence obtained from the current line of the mirGFF3 file, the *refseq* string is the output of the *adapt_refseq* function and is the reference sequence, finally, the *pos* integer is the reference position within the chromosome.

The output generates a list (*key_var*) which contains the variations found in the current lecture. There will be two types of variations, SNVs (variations between A, C, T and G bases) and Non SNVs (Indels). It also generates a list (in concordance of *key_var*) with the positions where the variants start (always in the reference sequence). The reference and alternate bases are literally the ones to include in their fields of the VCF format.

The function parses the expanded CIGAR to each of the two sequences (read and reference), and iterates finding the variations and its positions.

It is important to not repeat the same SNV, so a system is implemented in order to avoid it. For the SNVs the process is trivial, if the variation of the nucleotide is in the same position and is the same base, then it's the same SNV, but for indels it is a little more complex. There may be a Deletion in the same position as another lecture, which it can be interpreted as the same SNV, but that does not take into account that there may be more deletions (in succession), which is not the same case although the start position is. The system employed considers the previous nucleotide to check if there is a deletion (or an insertion in case of checking them) and marks it in the *key_var* list. For the Insertions it is trickier, because even if three insertions in succession are confirmed, the nucleotides involved may not be the same, therefore it is not the same SNV.

In that regard, the key of the variation will have the pattern of "DX" where X is the number of deletions in sequence and "IXIXIX…" for every insertion in the sequence, where X are the nucleotides inserted.

To clarify it, with the same CIGAR in others examples:

cigar_read = MMMMMMMMMMMMMMMMMMMIIIMM
cigar_ref = MMMMMMMMMMMMMMMMMMMMMDD
readseq = AAAAGCTGGGTTGAGAGCACGA
refseq = AAAAGCTGGGTTGAGAGGAGC
pos = 140926189

Then, the mapping goes this way:

```
MMMMMMMMMMMMMMMMMMMIIIMM
AAAAGCTGGGTTGAGAGCACGA

MMMMMMMMMMMMMMMMMMMMMDD
AAAAGCTGGGTTGAGAGGAGC
```

Underlined and in yellow is marked the start of the reference positions of the indels. Technically, it shouldn't be allowed to mark the position of the read sequence instead of the reference one, but both sequences are in sync because the code interprets the indels like this:

```
90123456789012345 6   7890     Reference position
XMMMMMMMMMMMMMMMMMMIIIMMDD     CIGAR extended
XAAAAGCTGGGTTGAGAGCACGA        Read Sequence
XAAAAGCTGGGTTGAGAG   GAGC      Reference Sequence
```

The green marks the start of the sequence, with position 140926189, in yellow it's shown the positions where the variant position in the reference starts (that is, 140926206 and 140926208).

The *key_var* coded are ICIAIC for the three insertions in a row and D2 for the two deletions. It is not needed to indicate the nucleotides deleted because they will always be the same (it's the reference sequence, not the read one).

Finally, the reference positions are marked in red and the alternate nucleotides in blue.

key_pos: [140926208, 140926206]
key_var: ['D2', 'ICIAIC'],
vcf_ref: ['AGC', 'G']
vcf_alt: ['A', 'GCAC']

Function	create_vcf
Input	mirgff3 (string), precursor (string), gtf (string), vcffile (string)
Output	Without return. Instead, a file is generated with the 'vcffile' name
Description	The most complex function of the script, generates a file with the VCF format

Table 11: create_vcf function

The function is called with four parameters, the name of the mirGFF3 file (mirgff3), the FASTA format sequences of all mature miRNA sequences (precursor) [2], the genome coordinates (gtf) [2] and the name of the output file.

The output is a .VCF file with the fields converted from the mirGFF3 format.

First, the function checks if the mirgff3 file exists, if it exists, decodes from UTF-8 codec with BOM (byte order mark) signature and encodes to UTF-8 and separates all the info by lines (ASCSII Linefeed (LF)) [3].

It follows by writing to the new .VCF file the meta-information lines. That is, the file format (version 4.3), fileDate (based on the system's date), source-ontology (based on the original mirGFF3 file), the INFO data (in this case "NS", the number of samples), the filters applied (REJECTED if filter not passed) and four lines (one per format type) with the Format IDs of TRC (total read counts), TSC (total SNV counts), TMC (total miRNA counts) and GT (genotype).

The headers are written after the meta-information, they are: CHROM (chromosome from the reference genome), POS (reference position), ID (self-explained) REF (reference base/s), ALT (alternate base/s), QUAL (quality, currently inactive because of no input data), FILTER (filter status), INFO (additional information) and one more for each sample.

An already existing function named *read_gff_line* returns the fields of the mirGFF3 format file. Its input is the mirGFF3 file. Another external function named *read_gtf_to_mirna* is used, this one returns a dictionary with information relative to the genome coordinates (in fact, this same file is the input of the function). The keys of this dictionary are actually the Parent attribute of the mirGFF3 format file, and the CHROM (chromosome), read sequence, reference positions, the CIGAR string, the variants (only the *iso_5p* is used), hairpin and miRNA are retrieved.

With the CIGAR string, the hairpin, the *iso_5p* variant and the read sequence the functions that find all the SNVs are called. A dictionary is created to save all the keys of SNVs to avoid duplicates. The information that contains this dictionary are all the fields needed in the VCF format and the key is formed by CHROM+POS+SNV.

Following the previous examples, for the SNV involving the two deletions, the key would be: **chrX-140926208-D2** (Chromosome X, reference position of 140926208 and D2 which corresponds to 2 deletions in a row). The ID selected is **hsa-miR-320d-nonSNP1294** which corresponds to the miRNA **hsa-miR-320d** and is the (particular) non SNV found number 1294. As explained previously, the reference and alternate bases are **AGC** and **A**, the quality is not needed so a **.** (dot) is placed. Because it has **pass** the filter this is its value. Finally, with 4 samples, the NS (number of samples) is **4**. The remaining fields of the dictionary are relative to every sample, but it is only needed for SNVs so they are not saved for non SNVs.

Another example, a SNV found in the position **140926210** of the chromosome **X** in the same miRNA as the previous example (hsa-miR-320d), an SNV is found (C->A). Because this is a SNV, the fields TRC, TSC, TMC and GT are needed to be informed for every sample.

The TRC value is equal to the sum of all difference sequences with the same SNV found, the TSC is the value of the expression of this SNV and the TMC the sum of all the SNVs found in the miRNA (hsa-miR-320d). The genotype (GT) will be 0|0 if there is no expression, 1|1 if the expression of the SNV equals to the expression of the miRNA, and 1|0 if not.
These fields were informed as:

TRC = [1,2,1,0]
TSC = [2,9,3,0]
TMC = [166, 1650, 507, 719]
GT = [1|0, 1|0, 1|0, 0|0]

Here we can say that there was no expression in the fourth sample and most expressed in the second sample for this SNV.

The lines generated for the non SNV and the SNV of the examples are as follows:

```
chrX   140926208   hsa-miR-320d-nonSNP1294       AGC   A       .
       Pass   NS=4

chrX   140926210   hsa-miR-320d-SNP10227 C       A       .       Pass
       NS=4 TRC:TSC:TMC:GT 1:2:166:1|0   2:9:1650:1|0 1:3:507:1|0
       0:0:719:0|0
```

3.2 Plots generated from the data

To analyze the data of the VCF file generated, a set of plots have been created. A list of miRNAs has been filtered before the computation of these plots, the complete list can be found in the appendix. This is based on the miRTop project that has defined the miRNAs that won't have cross-mapping events during alignment (sequences that map to multiple miRNAs with similar scores).

The first graph is a barplot that contains the number of SNVs in each sample:

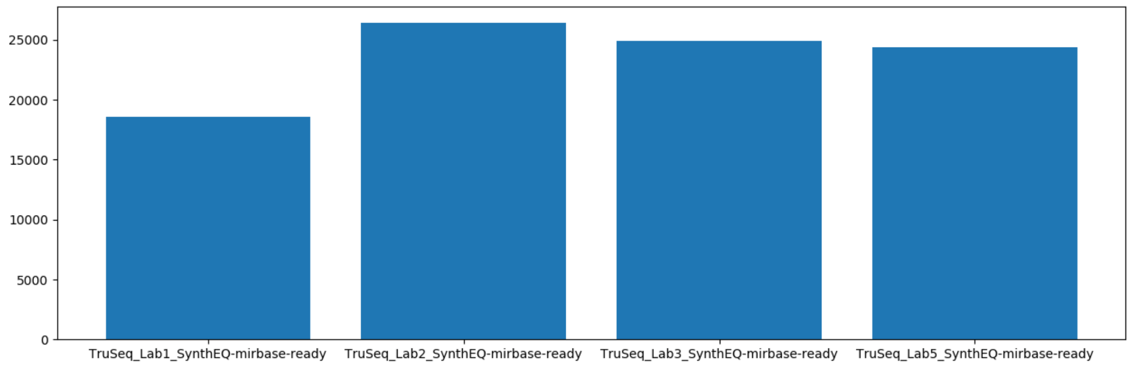


Figure 4: Barplot with number of SNVs per sample - Synthetic samples

The elevate number of these false positives SNVs is maybe due to the sequencing error rate since we assume the synthetic sample won't have any variation and only the sequence that was synthetized will be detected.

The second graph is a boxplot that contains the number of SNVs per miRNA and per sample:

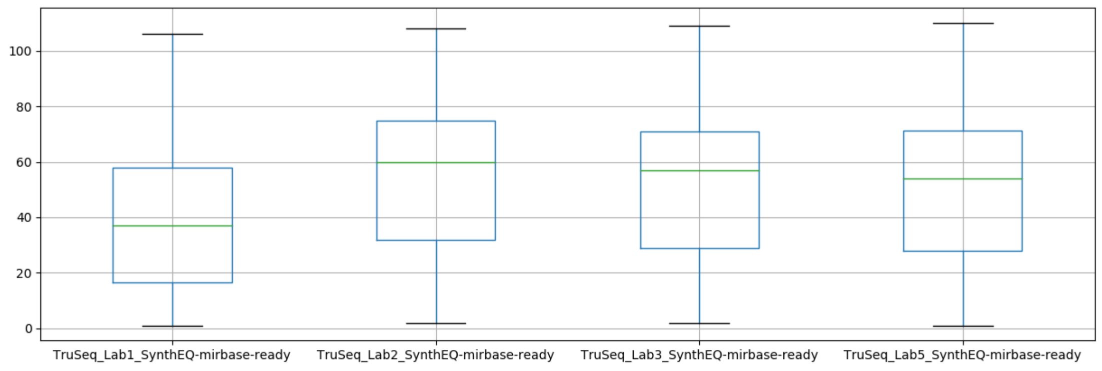


Figure 5: Boxplot with number of SNV per miRNA and sample - Synthetic samples

The number of SNVs per miRNA ranges from 0 to 100, having the most miRNAs between 20 and 60 SNVs. This is very interesting since the miRNAs in the synthetic sample should be equimolar (all miRNAs with the same abundance). Although the original work has confirmed that miRNAs are not detected at the same abundance, it was unknown the number of variations for each miRNA that the sequencing technology could generate. Here we confirm that there is a bias as well in the number of variants that each miRNAs have.

The third plot calculates a variable (importance) being. The expression of the SNV divided by the expression of the miRNA, in percentage (%).

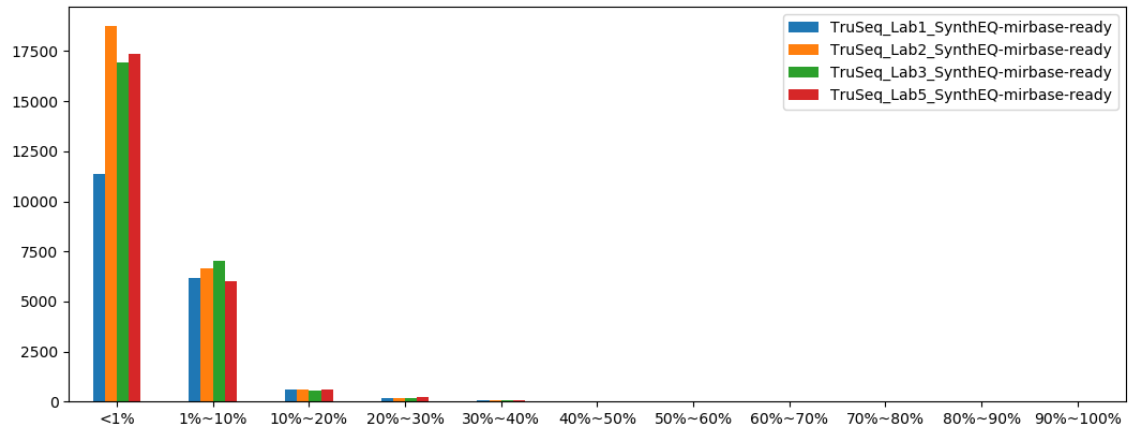


Figure 6: Importance of the SNVs versus the miRNA expression - Synthetic samples

The majority of the SNVs are 10% (or less) of the total miRNA abundance. The error sequencing rate for this technology has been described to be 0.1-1%. Although, almost 70% of SNVs seem to be under this value, still there are a high percentage that is over the expected rate. For instance, if we zoom in to observe the SNVs with an importance greater than 10%, we see less SNVs in proportion:

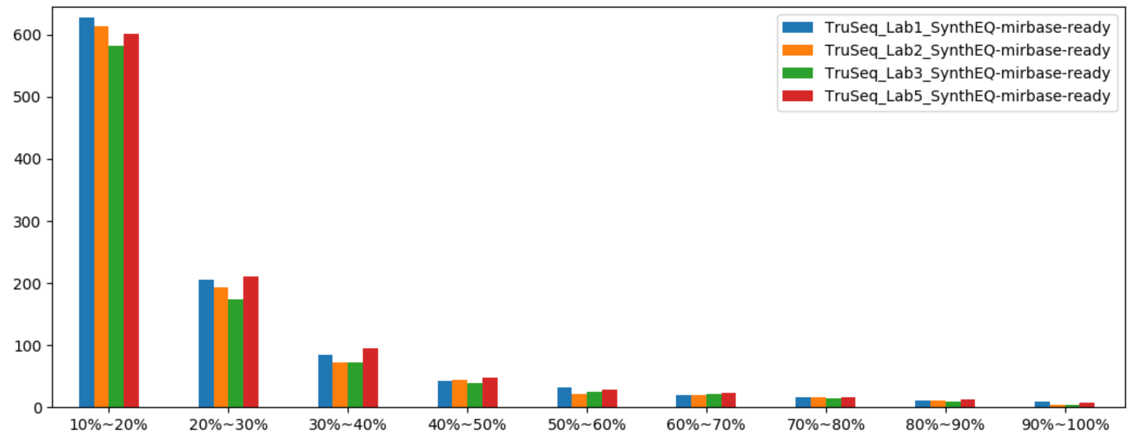


Figure 7: Importance of the SNVs versus the miRNA expression (zoomed in) - Synthetic samples

In comparative, the plots from real samples:

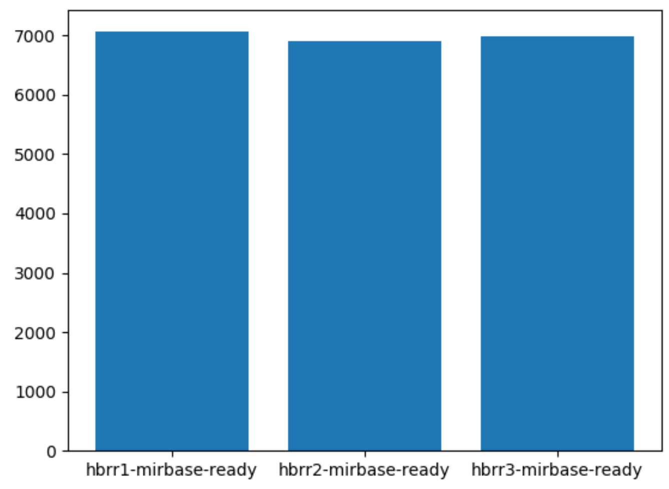


Figure 8: Barplot with number of SNVs per sample - Real samples

The number of SNVs have half the SNVs than the synthetic samples, with nearly 7.000 SNVs per sample.

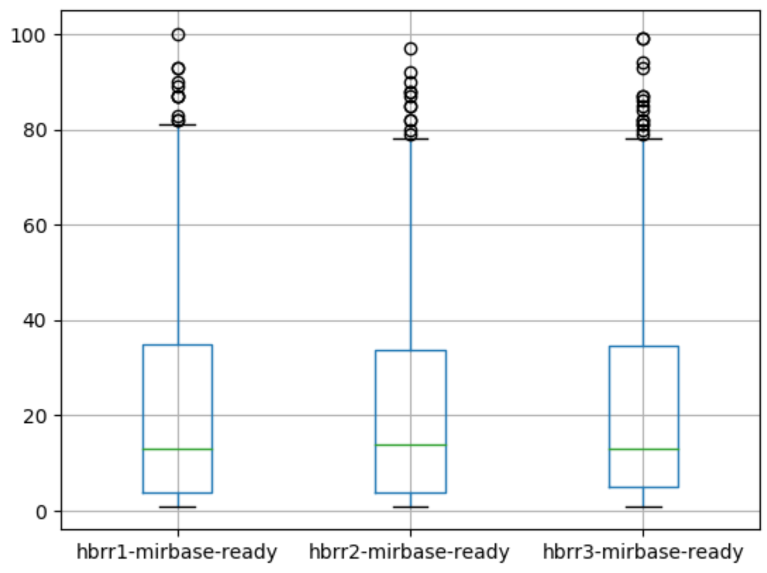


Figure 9: Boxplot with number of SNV per miRNA and per sample - Real samples

It is also seen in the boxplot, with only a few outliers, the expected number of SNVs in real samples is much lower than synthetic ones. This could be due to the fact that real samples have few miRNAs highly expressed and the rest are less represented.

The importance distribution follows a similar trend:

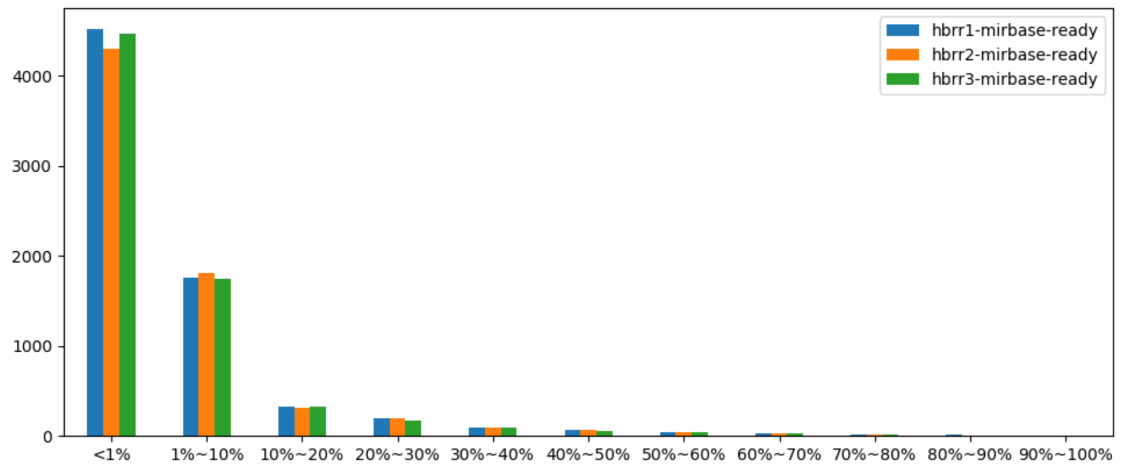


Figure 10: Importance of the SNVs versus the miRNA expression - Real samples

With a zoom in for SNVs with more than 10% of importance:

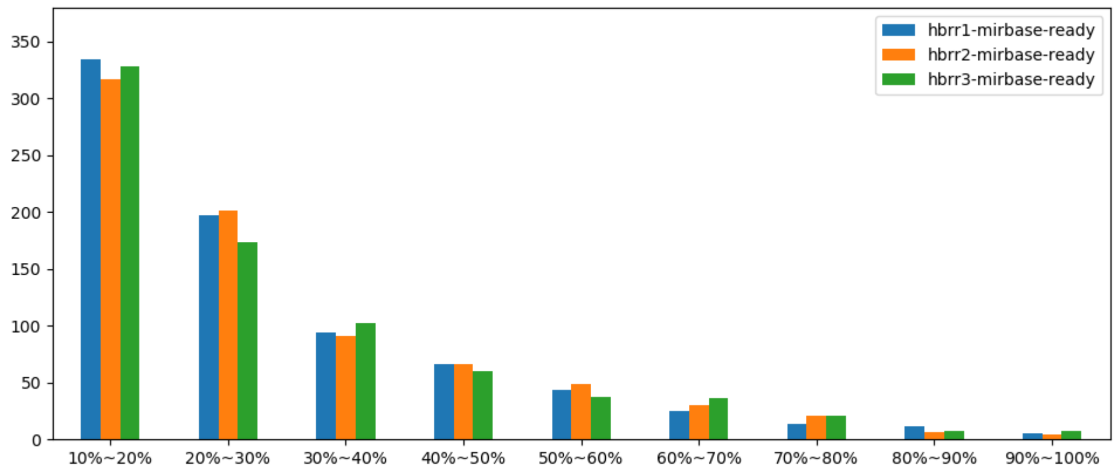


Figure 11: Importance of the SNVs versus the miRNA expression (zoomed in) - Real samples

The importance values behave in a similar way than synthetic samples, although it seems that the number of SNVs normalized by the total SNVs detected are higher in the importance range of 50% or more. This could be an indication that real samples could have an enrichment to real SNVs.

3.3 Results obtained

The results obtained after the conclusion of the project:

- The code that converts to VCF is implemented in mirtop: link to gitHub project
- It is a part of a paper to be published: link to the paper abstract
- Two VCF files have been generated from two mirGFF3 files that contained synthetic and real miRNA samples.

- General description of SVNs distribution among synthetic and real samples.

4. Conclusions

The main objectives set at the start of the project have been completed, a tool has been developed. Synthetic and real samples have been processed. A variety of different graphs with SNVs summaries were also generated.

This project has allowed to deepen knowledge of unfamiliar areas and to participate in a collaborative project that pursued the same goal.

The selected methodologies followed during the course of the project did not differ. Planning has been carried out as expected and adjustments to guarantee the success of the tasks did not alter the course.

Future lines of work may include more pipelines that uses the variant data of the format and statistics associated to them.

Because lack of time, another goal to pursue that has not been done is to perform an analysis which could involve machine learning algorithms with synthetic and real samples in order to obtain more precise information of how false positives could be filtered out. Although to reach this goal, there is a need to have a prior knowledge of validated SNVs in the samples, otherwise the machine learning algorithm will be bias to false positive SNVs. Another limitation of synthetic samples is that we are unsure about the purity of the sample, where some SNVs could be due to the error of synthesis process and not an error in the sequencing protocol.

5. Bibliography

[1] Cai X, Hagedorn CH, Cullen BR., "Human microRNAs are processed from capped, polyadenylated transcripts that can also function as mRNAs.," RNA. 2004 Dec;10(12):1957-66., 2004.

[2] Morin,R.D., O'Connor,M.D., Griffith,M., Kuchenbauer,F., Delaney,A., Prabhu,A.L., Zhao,Y., McDonald,H., Zeng,T., Hirst,M. et al., "Application of massively parallel sequencing to microRNA profiling and discovery in human embryonic stem cells.," Genome Res., 18, 610–621., 2008.

[3] Pantano, L., "GFF3 format with miRNA and isomiR information from sequencing data," [Online]. Available: https://github.com/miRTop/mirGFF3/blob/master/definition.md.

[4] Pantano, L., "project for small RNA standard annotations," [Online]. Available: http://mirtop.github.io/.

[5] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, 1000 Genomes Project Analysis Group, "The variant call format and VCFtools," Bioinformatics. 2011 Aug 1; 27(15): 2156–2158, 2011.

[6] M, Tewari, "Systematic assessment of next-generation sequencing for quantitative small RNA profiling: synthetic equimolar pool," 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE94584.

[7] T. B. P. R. K. R. H. G. Denli AM, "Processing of primary," Nature. 2004;432: 231–235.

[8] P. P. Perron MP, "Protein interactions and complexes in human microRNA biogenesis," Front Biosci. 2008;13: 2537–2547..

[9] B. DP, "Metazoan MicroRNAs," Cell. Elsevier; 2018;173: 20–51. .

[10] B. DP., "MicroRNAs: genomics, biogenesis, mechanism, and function.," Cell. 2004;116:281–297..

[11] N. M. Ardekani AM, "The Role of MicroRNAs in Human Diseases," Avicenna J Med.

[12] B. P. B. E. E. K. E. J. M. M. e. a. Desvignes T, "Nomenclature: A View Incorporating Genetic Origins, Biosynthetic Pathways, and," Trends Genet. 2015;31: 613–626.

[13] E. X. M. E. Pantano L, "SeqBuster, a bioinformatic tool for the processing and analysis of small RNAs datasets, reveals ubiquitous miRNA modifications in human embryonic cells.," Nucleic Acids Res. 2010;38: e34. .

[14] L. E. R. I. Loher P, "IsomiR expression profiles in human lymphoblastoid cell," Oncotarget. 2014;5: 8790–8802.

[15] R. I. Telonis AG, " Race Disparities in the Contribution of miRNA Isoforms and tRNADerived Fragments to Triple-Negative Breast Cancer.," Cancer Res. 2018;78: 1140–1154.

[16] T. A. L. P. L. E. R. I. Magee RG, " Profiles of miRNA Isoforms and tRNA Fragments in Prostate Cancer.," Sci Rep. 2018;8: 5314. .

[17] M. R. L. P. C. I. L. E. R. I. Telonis AG, "Knowledge about the presence or absence of miRNA isoforms (isomiRs) can successfully discriminate amongst 32 TCGA cancer types," Nucleic Acids Res. 2017;45: 2973–2985. .

[18] D. P. L. K. E. J. M. G. U. B. F. J. S. E. A.-P. V. B. R. E. V. O. P. G. T. E. F. M. R. F. e. a. Lorena Pantano, "Unification of miRNA and isomiR research: the mirGFF3 format and the mirtop API," 2018.

[19] F. &. G. C. &. B. M. &. H. K. &. B. M. &. S. J. &. M. G. Pfeiffer, "Systematic evaluation of error rates and causes in short samples in next-generation sequencing," Scientific Report, svolume 8, Article number: 10950, 2018.

[20] [Online]. Available: http://gmod.org/wiki/GFF3. [Accessed 19 October 2018].

[21] [Online]. Available: http://gmod.org/wiki/GMOD_Components. [Accessed 19 October 2018].

[22] G. P. A. A. E. F. Gianvito Urgese, "isomiR-SEA: an RNA-Seq analysis tool for miRNAs/isomiRs expression level profiling and miRNA-mRNA interaction sites evaluation," BMC BioinformaticsBMC series – open, inclusive and trusted201617:148, 31 March 2016.

[23] [Online]. Available: https://samtools.github.io/hts-specs/SAMv1.pdf. [Accessed 21 October 2018].

[24] [Online]. Available: https://samtools.github.io/hts-specs/VCFv4.3.pdf. [Accessed 21 October 2018].

[25] O. M. G. M. K. F. D. A. P. A.-L. e. a. Morin RD, "Application of massively parallel sequencing to microRNA profiling and discovery in human," Genome Res. 2008;18: 610–621..

6. Appendices

6.1 Script to_vcf.py

The script that reads a mirGFF3 file and generates a VCF one.

```python
from __future__ import print_function

from fasta import read_precursor
from mapper import read_gtf_to_precursor, read_gtf_to_mirna
from body import read_gff_line

import time
import datetime
import sys

def cigar_length(cigar):
    """
    Args:
        'cigar(str)': CIGAR standard of a compressed alignment representation, this CIGAR omits the '1' integer.
    Returns:
        'total_n(int)': CIGAR length in nucleotides.
    """
    total_n = 0
    match_n = "0"
    for i in cigar:
        if i.isdigit():
            match_n = match_n + str(i)
        else:
            total_n = total_n + int(match_n) + 1
            if i == "D" or (i == "M" and match_n != "0"):
                total_n = total_n - 1
            match_n = "0"
    return(total_n)

def cigar_2_key(cigar_read, cigar_ref, readseq, refseq, pos):
    """
    Args:
        'cigar_read(str)': CIGAR extended string of the read sequence, output of the expand_seqs.
        'cigar_ref(str)': CIGAR extended string of the reference sequence, output of the expand_seqs.
        'readseq(str)': the read sequence
        'refseq(str)': the reference sequence
        'pos(str)': the initial current position of the chromosome.
    Returns:
        'key_pos(str list)': a list with the positions of the variances.
        'key_var(str list)': a list with the variant keys found.
        'ref(str list)': reference base(s).
        'alt(str list)': altered base(s).
    """
    key_pos = []
    key_var = []
    ref = []
    alt = []
    n_I = 0  # To balance the position between read and ref sequences
    for i in range(len(cigar_ref)):  # Parsing for SNPs and Dels
        if cigar_ref[i] == "M":
            continue
        elif cigar_ref[i] in ["A", "T", "C", "G"]:
            key_pos.append(pos + i+1 + n_I)
            key_var.append(cigar_ref[i])
            ref.append(refseq[i])
            alt.append(cigar_ref[i])
        elif cigar_ref[i] == "D":
            if i == 0:
                print("Unexpected 'D' in the first position of CIGAR")
            elif i > 0:
                if cigar_ref[i-1] == "D":
                    ref[-1] = ref[-1] + refseq[i]  # Adds new Del in the REF column
                    key_var[-1] = "D" + str(int(key_var[-1][1:]) + 1)  # Adds 1 to the number of Dels in succession
                else:
                    key_pos.append(pos + i + n_I)
                    key_var.append("D1")
                    ref.append(refseq[i-1:i+1])
                    alt.append(refseq[i-1])

    for i in range(len(cigar_read)):  # Parsing for Insertions
        if cigar_read[i] == "I":
            if i == 0:
                print("Unexpected 'I' in the first position of CIGAR")
            elif cigar_read[i-1] == "I":
                alt[-1] = alt[-1] + readseq[i]  # Adds the new Insert in the ALT column
                key_var[-1] = key_var[-1] + 'I' + readseq[i]  # Adds new Ins in the Key
                n_I = n_I - 1
            else:
                key_pos.append(pos + i + n_I)
                alt.append(readseq[i-1:i+1])
                ref.append(readseq[i-1])
                key_var.append("I" + alt[-1][-1])
                n_I = n_I - 1
        else:
            continue
    return (key_pos, key_var, ref, alt)

def expand_seqs(cigar):
    n_Mpar = "0"
    cigar_exp = ""
    for i in cigar:
        if i.isdigit():
            n_Mpar = n_Mpar + i  # Gets the number of matched nucleotides
        elif i == "M":
            if n_Mpar == "0":
                cigar_exp = cigar_exp + "M"
            else:
                cigar_exp = cigar_exp + str(''.join("M"*int(n_Mpar)))  # Adds all the "M"s
                n_Mpar = "0"
        elif i in ["A", "T", "C", "G", "I", "D"]:
            cigar_exp = cigar_exp + i
        else:
            print("Unexpected value")
    cigar_exp_read = cigar_exp.replace("D", "")
    cigar_exp_ref = cigar_exp.replace("I", "")
    return(cigar_exp_read, cigar_exp_ref)


def adapt_refseq(cigar_ref, hairpin, parent_ini_pos, var5p):
    index = parent_ini_pos + var5p
    max_index = index + len(cigar_ref)
    refseq = hairpin[index:max_index]
    return(refseq)


def create_vcf(mirgff3, precursor, gtf, vcffile):
    """
    Args:
        'mirgff3(str)': File with mirGFF3 format that will be converted

        'precursor(str)': FASTA format sequences of all miRNA hairpins
        'gtf(str)': Genome coordinates
        'vcffile': name of the file to be saved
    Returns:
        Nothing is returned, instead, a VCF file is generated
    """
    #Check if the input files exist:
    try:
        gff3_file = open(mirgff3, "r")
    except IOError:
        print ("Can't read the file", end=mirgff3)
        sys.exit()
    with gff3_file:
        data = gff3_file.read().decode("utf-8-sig").encode("utf-8")

    gff3_data = data.split("\n")
    vcf_file = open(vcffile, "w")

    ver = "v4.3"  # Current VCF version formatting
    vcf_file.write("##fileformat=VCF%s\n" % ver)
    date = datetime.datetime.now().strftime("%Y%m%d")
    vcf_file.write("##fileDate=%s\n" % date)
    source = "\n".join(s for s in gff3_data if "## source-ontology: " in s)[20:]
    line = 0
    sample_names = []
    while gff3_data[line][:2] == "##":
        if gff3_data[line][:19] == "## source-ontology:":
            source = gff3_data[line][20:]
        elif gff3_data[line][:11] == "## COLDATA:":
            sample_names = gff3_data[line][12:].split(",")
        line += 1
    vcf_file.write("##source=%s\n" % source)
    # ref_file = "N/A"  # Temporary
    # vcf_file.write("##reference=%s\n" % ref_file)
    vcf_file.write('##INFO=<ID=NS,Type=Integer,Description="Number of samples"\n')
    vcf_file.write("##FILTER=<ID=REJECT,Description='"'Filter not passed'"'>\n")
    vcf_file.write('##FORMAT=<ID=TRC,Number=1,Type=Integer,Description="Total read count">\n')
    vcf_file.write('##FORMAT=<ID=TSC,Number=1,Type=Integer,Description="Total SNP count">\n')
    vcf_file.write('##FORMAT=<ID=TMC,Number=1,Type=Integer,Description="Total miRNA count">\n')
    vcf_file.write('##FORMAT=<ID=GT,Number=1,Type=Integer,Description="Genotype">\n')
    header = "#CHROM\tPOS\tID\tREF\tALT\tQUAL\tFILTER\tINFO\tFORMAT"
    for s in range(len(sample_names)):
        header = header + "\t" + sample_names[s]
    vcf_file.write(header)

    hairpins = read_precursor(precursor)
    gff3 = read_gtf_to_precursor(gtf)
    gtf_dic = read_gtf_to_mirna(gtf)
    all_dict = dict()  # initializing an empty dictionary where all info will be added
    key_list = []  # Initializing a list which will contain all the keys of the dictionary
    mirna_dict = dict()  # initializing an empty dictionary where mirna info will be put
    n_SNP = 0
    n_noSNP = 0
    no_var = 0
    for line in range(0, len(gff3_data)):
        if gff3_data[line][1] == "#":
            continue
        else:
            gff_fields = read_gff_line(gff3_data[line])
            gtf_name = gff_fields['attrb']['Name']
            gtf_parent = gff_fields['attrb']['Parent']
            if gtf_parent not in gff3:
                continue
            if gtf_name not in gff3[gtf_parent]:
                continue
            parent_ini_pos = gff3[gtf_parent][gtf_name][0]
            parent_end_pos = gff3[gtf_parent][gtf_name][1]
            vcf_chrom = gtf_dic[gtf_name][gtf_parent][0]
            vcf_pos = int(gff_fields['start']) + int(gtf_dic[gtf_name][gtf_parent][1])
            hairpin = hairpins[gtf_parent]
            variants = gff_fields['attrb']['Variant'].split(",")
            cigar = gff_fields['attrb']["Cigar"]
            readseq = gff_fields['attrb']['Read']

            var5p = [s for s in variants if 'iso_5p' in s]  # Obtaining iso_5p value:
            if len(var5p):
                var5p = int(var5p[0][7:])  # Position of iso_5p value
            else:
                var5p = 0  # 0 if iso_5p is not found

            (cigar_exp_read, cigar_exp_ref) = expand_seqs(cigar)
            refseq = adapt_refseq(cigar_exp_ref, hairpin, parent_ini_pos, var5p)
            (key_pos, key_var, vcf_ref, vcf_alt) = cigar_2_key(cigar_exp_read, cigar_exp_ref, readseq, refseq,
                                                               (vcf_pos + var5p))

            if len(key_var) > 0:
                for s in range(len(key_var)):
                    key_dict = vcf_chrom + '-' + str(key_pos[s]) + '-' + str(key_var[s])
                    raw_counts = gff_fields['attrb']['Expression']
                    raw_counts = [int(i) for i in raw_counts.split(',')]
                    nozero_counts = [int(i > 0) for i in raw_counts]  # counts for every sample if expr != 0.
                    if str(key_var[s]) in ["A", "C", "T", "G"]:
                        if gtf_name in mirna_dict:  # Adding expression values to same mirnas
                            mirna_dict[gtf_name]['Z'] = [sum(x) for x in zip(mirna_dict[gtf_name]['Z'], raw_counts)]
                        else:
                            mirna_dict[gtf_name] = {}
                            mirna_dict[gtf_name]["Z"] = raw_counts
                    if key_dict in all_dict:
                        if all_dict[key_dict]["Type"] in ["A", "C", "T", "G"]:
                            all_dict[key_dict]['X'] = [sum(x) for x in zip(all_dict[key_dict]['X'], nozero_counts)]
                            all_dict[key_dict]['Y'] = [sum(x) for x in zip(all_dict[key_dict]['Y'], raw_counts)]
                    else:
                        key_list.append(key_dict)
                        all_dict[key_dict] = {}
                        all_dict[key_dict]["Chrom"] = vcf_chrom
                        all_dict[key_dict]["Position"] = key_pos[s]
                        all_dict[key_dict]["mirna"] = gtf_name
                        all_dict[key_dict]["Type"] = key_var[s]
                        if key_var[s][0] in ["A", "C", "T", "G"]:
                            n_SNP += 1
                            all_dict[key_dict]["SNP"] = True
                            all_dict[key_dict]["ID"] = gff_fields['attrb']['Name'] + '-SNP' + str(n_SNP)
                            all_dict[key_dict]['X'] = nozero_counts
                            all_dict[key_dict]['Y'] = raw_counts
                        else:
                            n_noSNP += 1
                            all_dict[key_dict]["SNP"] = False
                            all_dict[key_dict]["ID"] = gff_fields['attrb']['Name'] + '-nonSNP' + str(n_noSNP)
                        all_dict[key_dict]["Ref"] = vcf_ref[s]
                        all_dict[key_dict]["Alt"] = vcf_alt[s]
                        all_dict[key_dict]["Qual"] = "."
                        all_dict[key_dict]["Filter"] = gff_fields['attrb']['Filter']
                        all_dict[key_dict]["Info"] = "NS=" + str(len(sample_names))
            else:
                no_var += 1
    #  Writing the VCF file:
    for s in key_list:
        variant_line = ("\n%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s" %
                       (all_dict[s]["Chrom"], all_dict[s]["Position"], all_dict[s]["ID"],
                        all_dict[s]["Ref"], all_dict[s]["Alt"], all_dict[s]["Qual"],
                        all_dict[s]["Filter"], all_dict[s]["Info"]))
        if all_dict[s]["Type"] in ["A", "T", "C", "G"]:
            format_col = "TRC:TSC:TMC:GT"
            variant_line = variant_line + "\t" + format_col
            samples = ""
            for n in range(len(sample_names)):
                X = all_dict[s]["X"][n]
                Y = all_dict[s]["Y"][n]
                Z = mirna_dict[all_dict[s]["mirna"]]["Z"][n]
                if Y == 0:
                    GT = "0|0"
                elif Z == Y:
                    GT = "1|1"
                else:
                    GT = "1|0"
                samples = samples + "\t" + str(X) + ":" + str(Y) + ":" + str(Z) + ":" + GT
            variant_line = variant_line + samples
        else:
            format_col = ""
            variant_line = variant_line + format_col
        vcf_file.write(variant_line)
    vcf_file.close()
```

6.2 Script plot_vcf_graphs.py

Adapted to plot and get 3 samples for real data. The code for the synthetic data is slightly modified:

```python
from __future__ import print_function
import sys, csv
import matplotlib.pyplot as plt
import pandas as pd

def SNP_to_dict(line, n_samples, snpdict):
    """

    """
    key = line[2]  # ID
    snpdict[key] = {}
    snpdict[key]['mirna'] = key[:(key.find("-SNP"))]  # Gets the mirna name string
    snpdict[key]['ID'] = line[0] + '-' + line[1] + '-' + line[4]  # CHROM + POS + ALT
    samples_data = line[len(line) - n_samples:]
    samples = []  # Expression values will be appended here
    for sample in samples_data:
        sample = sample.replace(':', ',')
        samples.append(sample[:len(sample)].split(","))
    snpdict[key]['X'] = []
    snpdict[key]['Y'] = []
    snpdict[key]['Z'] = []
    snpdict[key]['GT'] = []
    for i in range(n_samples):
        snpdict[key]['X'].append(samples[i][0])
        snpdict[key]['Y'].append(samples[i][1])
        snpdict[key]['Z'].append(samples[i][2])
        snpdict[key]['GT'].append(samples[i][3])
    return(snpdict)


def create_vcf_tables(mirvcf, mirna_list_file):
    """
    Args:
        'mirvcf': a file with VCF format
        'mirna_list_file': a .csv list which contains the miRNAs to analyze
    Returns:
        ''
    """
    mirna_list=[]
    with open(mirna_list_file, 'rb') as csvfile:
        reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in reader:
            mirna_list.append(', '.join(row))
    mirna_list = (mirna_list[1:])

    # Check if the input file exist:
    try:
        vcf_file = open(mirvcf, "r")
    except IOError:
        print ("Can't read the file", end=mirvcf)
        sys.exit()
    with vcf_file:
        data = vcf_file.read().decode("utf-8-sig").encode("utf-8")

    vcf_data = data.split("\n")  # Parsing the VCF file:
    mirna_SNPs = []
    mirna_keys = []
    snp_dict = {}
    mirna_snp_dict = {}
    for line in range(0, len(vcf_data)):
        if vcf_data[line][0:2] == "#C":
            sample_names = vcf_data[line].split("\t")[9:]  # Get sample names from header
        elif vcf_data[line][0] == "#":
            continue
        else:
            variant = vcf_data[line]
            cols_info = variant.split("\t")
            # print("cols_info: ", cols_info)
            n_samples = int(cols_info[7][3:])
            mirna = cols_info[2]
            mirna_dict = dict()

            if mirna.find("-SNP") == -1:  # Non-SNP
                continue
            else:
                mirna = mirna[:(mirna.find("-SNP"))]
                if mirna in mirna_list:
                    snp_dict = SNP_to_dict(cols_info, n_samples, snp_dict)
                    samples_data = cols_info[len(cols_info) - n_samples:]
                    samples = []  # Expression values will be appended here
                    for sample in samples_data:
                        sample = sample.replace(':', ',')
                        samples.append(sample[:-2].split(","))
                    SNPs_samples = [0] * n_samples  # Creating an empty list to allocate number of diff SNP
                    SNPs_same = [0] * n_samples  # Creating an empty list to allocate number of same SNPs


    snp_df = pd.DataFrame.from_dict(snp_dict, orient='index')

    ### INI PLOT 3 ###
    cols = ["isomiR ID", "X", "Y", "Z", "Importance", "Group"]
    cols = ["Importance"]
    count_list = []
    counts_list = []
    for j in range(3):
        lst = []
        for i in range(len(snp_df)):
            if int(snp_df["X"][i][j]) == 0:
                continue
                  elif int(snp_df["Z"][i][j]) == 0:
                      importance = 0
                  else:
                      importance = int(snp_df["Y"][i][j]) / float(snp_df["Z"][i][j]) * 100
                  if importance < 1:
                      imp_group = 0
                  else:
                      imp_group = (int(importance)/10)+1  # To classify
                      if imp_group > 10: imp_group = 10
                  lst.append([imp_group])
              count_list.append(lst)
              plot2_df = pd.DataFrame(count_list[j], columns=cols)
              counts_list.append(plot2_df['Importance'].value_counts().sort_index().tolist())

          index = ["<1%", "1%~10%", "10%~20%", "20%~30%", "30%~40%", "40%~50%",
                   "50%~60%","60%~70%","70%~80%","80%~90%", "90%~100%"]

          counts__df = pd.DataFrame({sample_names[0]:counts_list[0],
                                     sample_names[1]:counts_list[1],
                                     sample_names[2]:counts_list[2]},
                                     index = index)
          counts__df.plot.bar(rot=0)
          plt.show()

          ### END PLOT 3 ###

          ### INI PLOT 1 ###

          SNPs_count = []
          SNPss_count = []
          mirnas = []
          mirnass = []

          for j in range(3):
              SNPs_count = []
              mirnas = []
              for i in range(len(snp_df)):
                  if int(snp_df["X"][i][j]) > 0:
                      SNPs_count.append(1)
                      mirnas.append(snp_df["mirna"][i])
              SNPss_count.append(SNPs_count)
              mirnass.append(mirnas)

          plt.bar(sample_names[0:3], [len(mirnass[0]), len(mirnass[1]),
                                      len(mirnass[2])])
          plt.show()
          ### END PLOT 1 ###

          ### INI PLOT 2 ###

          plot2_s_list = []
          for j in range(3):
              plot2_s1 = pd.DataFrame({"mirna": mirnass[j], "SNPs": SNPss_count[j]})
              plot2_s1_sum = plot2_s1.groupby(["mirna"]).sum()
              plot2_s_list.append(plot2_s1_sum)

          data = pd.DataFrame({sample_names[0]:plot2_s_list[0]['SNPs'],
                               sample_names[1]:plot2_s_list[1]['SNPs'],
                               sample_names[2]:plot2_s_list[2]['SNPs']})
          data.boxplot()
          plt.show()

          ### END PLOT 2 ###
```