# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR D´ENGINYERIA AGRONÒMICA I DEL MEDI NATURAL

# *ANÁLISIS DE DATOS DE GENOTIPADO A GRAN ESCALA*

TRABAJO FIN DE GRADO EN BIOTECNOLOGÍA

ALUMNO/A: MARTÍNEZ GIMÉNEZ, FELIPE

TUTOR/A: BLANCA POSTIGO, JOSÉ MIGUEL
TUTOR ACADÉMICO: CAÑIZARES SALES, JOAQUÍN

***Curso Académico: 2014-2015***

**VALENCIA, FECHA**

| | |
|---|---|
| Título: | ANÁLISIS DE DATOS DE GENOTIPADO A GRAN ESCALA |
| Autor del TFG: | Alumno: D. Felipe Martínez Giménez |
| Localidad y fecha: | Valencia, septiembre de 2015 |
| Tutor académico: | Prof. D.Joaquín Cañizares Sales |
| Tutor: | D. José Miguel Blanca Postigo |


**Resumen**

Las nuevas tecnologías de secuenciación y genotipado han abaratado el coste de estos análisis y propiciado el desarrollo de proyectos que contemplan el análisis de cientos o miles de individuos. Además, estas tecnologías son capaces de obtener el genotipo de millones marcadores genéticos en cada muestra. La combinación de número de individuos y número de marcadores crean matrices de datos inmensas que no son fáciles de analizar y manejar. Gran parte del software de análisis genéticos existentes no está adaptado al manejo de esta cantidad de datos, haciendo necesario el desarrollo de nuevos programas o la adaptación de los ya existentes a esta cantidad de información. Por otro lado, el manejo de esta cantidad de datos también resulta complejo para el investigador, por lo que es necesario el desarrollo de herramientas que faciliten su manejo, almacenamiento y visualización. El objetivo de este proyecto se enmarca en este problema: el manejo y la visualización de matrices con información genéticas de cientos de individuos procedentes de las nuevas herramientas de genotipado. Es necesario agilizar la gestión y el acceso a esta multitud de datos para poder realizar los diferentes análisis genéticos. Concretamente en este proyecto se van a estudiar diferentes estrategias y paquetes del lenguaje Phython para facilitar el acceso a los datos de genotipado y utilizar estos para realizar análisis multivariantes.



**Palabras claves**

Genotipado, bioinformática, Python, computación altas prestaciones

**Abstract**

New sequencing and genotyping technologies have cheapened the cost of these analyzes and led the development of projects involving the analysis of hundreds or thousands of individuals. Moreover, these technologies are able to get genotype million genetic markers in each sample. The combination of number of individuals and number of markers create huge data arrays, that aren't easy to analyze and manage. Much of the existing genetic analysis software isn't adapted to manage this amount of data, necessitating the development of new programs or adapting existing ones to this wealth of information. On the other hand, the management of this amount of data is also complex for researchers, so that the development of tools to facilitate its handling, storage and visualization is necessary. The objective of this project is part of this problem: handling and display matrices with genetic information from hundreds of individuals from new tools of genotyping. It's necessary to streamline the management and access to this multitude of data to perform the different genetic analysis. Specifically in this project they are to study different strategies and Python language packages for easy access to genotyping data and use these to perform multivariate analysis

**Key Words**

Genotyping, bioinformatics, Python, high performance computation

**ÍNDICE**

# Nomenclaturas y abreviaturas

ACP: Análisis de components principales

af: allele frequency

DNA: Deoxyribonucleic acid

GiB: Gibibyte

GB: Gigabyte

Gb: Gigabases

GBS: Genotyping By Sequencing

HDF5: Hierarchical Data Format 5

jpg: (JPEG) Joint Photographic Experts Group

maf: maximum allele frequency

MB: Megabytes

Mb: Megabases

NASA: National Aeronautics and Space Administration

np: NumPy

NumPy: Numeric Python

png: portable network graphics

s: segundos

SNP: Single Nucleotide Polymorphism

SO: Sistema Operativo

VCF: Variant Call Format

TB:Terabyte

# 1.INTRODUCCIÓN

## 1.1. Desarrollo de los sistemas de secuenciación

### 1.1.1 Los inicios

Fredrerik Sanger diseño en 1975 el método de secuenciación que acabaría recibiendo su nombre. Concretamente, en este método se aísla el DNA a secuenciar y se copia usando oligonucleótidos sintéticos, una DNA-polimerasa, los cuatro tipos de nucleótidos, encontrándose uno de ellos marcado radiactivamente y un dideoxinucleótido (se realiza el proceso en 4 tubos distintos, uno por cada dideoxinucleótido). De este modo la polimerasa une el nucleótido complementario a la cadena molde hasta que llega un dideoxinucleótido. En la reacción se generaban fragmentos marcados radiactivamente de distintos tamaños (dependiendo de cuándo se había incorporado el dideoxinucleótido que interrumpía la polimerización).

Tras la reacción se desnaturaliza el DNA y se coloca cada tubo en un carril de un gel de acrilamida y corre la electroforesis. Por último se obtiene una radiografía de la cual se puede leer la secuencia pues el último nucleótido será el marcado en cada reacción/carril y la longitud de la secuencia y por tanto posición de esa base está ligada a su posición en el gel (Sanger, 1975).

A modo de ejemplo en la figura1, tendríamos la radiografía del gel del cual se puede leer la secuencia desde las bandas más alejadas (son las que más han corrido al ser más pequeñas) hasta la más cercana, sabiendo que nucleótido radiactivo hay en cada carril.

La secuencia es: AGCCGTA

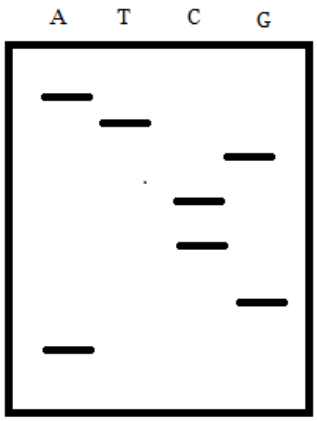


Figura1: Esquema de un gel para secuenciar por el método de Sanger

Esta técnica fue incluyendo mejoras tales como sustituir el marcaje radioactivo por fluoróforos, la automatización de la lectura de las bandas o la sustitución de los geles de electroforesis por electroforesis capilar. Los secuenciadores diseñados por Applied Biosystems podían llegar a tener hasta 96 capilares, lo que le permite secuenciar cerca de 96kb en 3 horas. (Morozova, et al 2008)

### 1.1.2 Segunda generación de secuenciadores

A partir del año 2004 aparecieron distintos métodos de secuenciación nuevos. Roche, Illumina y Applied Biosystems. En líneas generales el funcionamiento de estos secuenciadores es el siguiente:

Las tres se basan en la reacción de la PCR.

-Roche: Su secuenciador (tecnología 454) se conoce también como pirosecuenciación. En primer lugar se prepara el DNA fragmentándolo y uniéndole a los fragmentos generados unos adaptadores que servirán como secuencias conocidas para la PCR. La PCR se hace en una emulsión de aceite y esferas sintéticas, lo que permite realizar millones de reacciones de PCR en paralelo. Al finalizar la PCR aquellas esferas con DNA tendrán hasta 10 millones de copias de uno de los fragmentos de DNA. Las esferas se colocan sobre una placa con pocillos de tamaño tal que por azar en casi todos los pocillos haya una esfera. La reacción de secuenciación, se realiza inyectando una solución con uno de los 4 nucleótidos, la DNA-polimerasa, APS y luciferina. Si se une el nucleótido se liberará PPi que reaccionará con el APS obteniéndose ATP que será usado por la luciferasa para producir luz, la cual será detectada por una cámara de alta resolución.

Una vez obtenida la señal, se lavan los reactivos y se inicia un nuevo ciclo: se inyecta de nuevo la solución, pero, en esta ocasión, incluyendo un nucleótido distinto. Sabiendo qué nucleótido ha inyectado en cada ronda y teniendo una imagen de que pocillos que se iluminan en cada ronda, el secuenciador obtiene las secuencias de todos sus puntos (Patrick 2007).

-Illumina: Su secuenciador utiliza nucleótidos marcados fluorescentemente, los cuales impiden que siga la reacción. La marca fluorescente, que impide que la reacción de polimerización continúe es reversible, de modo que una vez eliminado el fluoróforo la reacción puede continuar. Para comenzar se fragmenta el DNA al azar y se le ligan adaptadores en ambos extremos. Estos fragmentos se unen a una placa que tiene los cebadores de la reacción PCR unidos. La PCR se realiza en la placa de modo que donde se ancla cada fragmento inicialmente se generarán los productos de reacción. Llegados a este punto se suceden rondas en las cuales se inyectan los cuatro nucleótidos marcados con un fluoróforo distinto cada uno, de modo que al estar marcados solo uno se unirá en cada caso, se tomará una foto y se eliminara el fluoróforo durante el lavado.(ILLUMINA, 2015).

En el mismo documento donde explican este funcionamiento podemos ver una gráfica (figura 2) que muestra cómo ha ido evolucionando su tecnología.

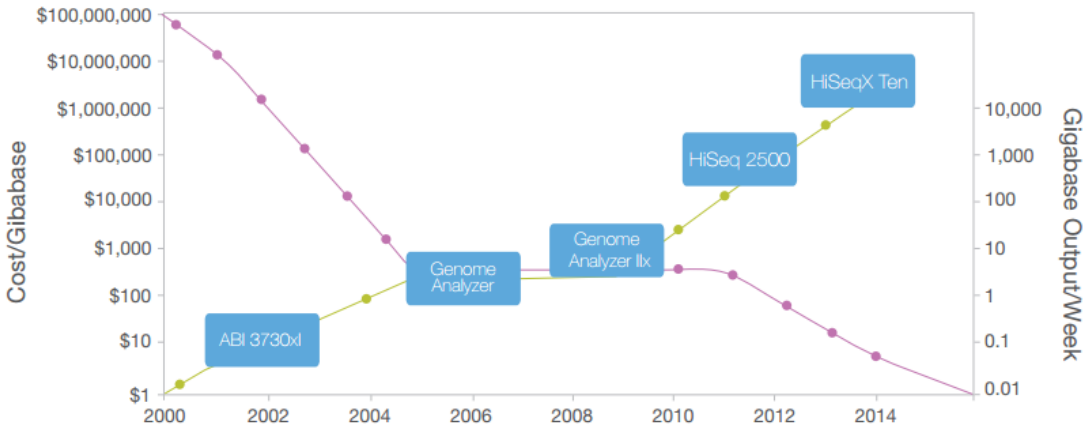


Figura2: Gráfica de Illumina de la evolución del coste/gigabase y gigabases/semana de secuenciación

-Applied Biosystems: Desarrolló un secuenciador (tecnología SOLiD) que se basa en la hibridación y ligación de oligonucleótidos. Estos oligonucleótidos poseen dos nucleótidos normales, tres núcleotidos degenerados y tres inosinas que se unen a cualquier nucleótido, estando la última marcada con un color según que pareja de nucleótidos haya al principio del modo que se puede ver en la figura3.La manera de llegar a los fragmentos que se van a secuenciar es similar a la tecnología de 454. En cada ronda se lanzan oligonucleotidos con todas las combinaciones posibles junto a una ligasa que ligará el oligo que hibride con la cadena, luego se lava y toma una imagen del color. Tras esto se eliminan parte de los nucleótidos degenerados y las inosinas quedando solo los dos primeros nucleotidos y los tres degenerados. Se repite el proceso. Una vez finalizado se desnaturaliza el DNA y se vuelve a realizar el proceso empezando por una base más adelante. Una vez se tienen las imágenes de las dos lecturas, se va deduciendo que bases son en función de la combinación de colores. Por ejemplo si las tres primeras bases fueran ATG en la primera imagen de la primera lectura se habría obtenido el color rojo y en la primera imagen de la segunda lectura el verde. Este secuenciador puede llegar a 1-3Gb en 8 días. (Morozova, et al 2008)

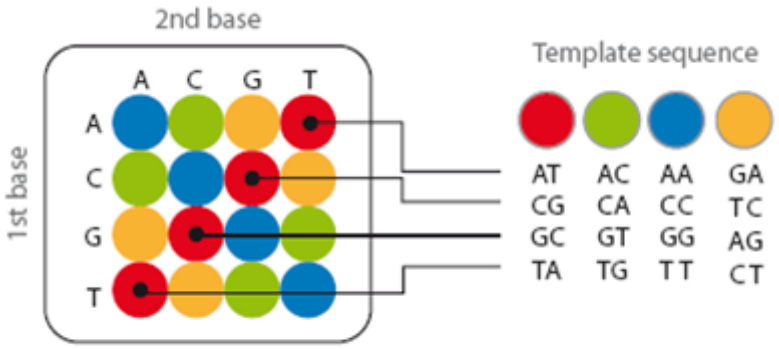


Figura 3: Ejemplo de tabla de correspondencia de colores y parejas de nucleótidos.

### 1.1.3 Tercera generación de secuenciadores

Una de las grandes características de esta generación es la secuenciación de una única molécula de DNA, sin la necesidad de amplificarla como en las de segunda generación.

De estos nuevos métodos, podemos poner dos ejemplos de empresas que venden estos secuenciadores. El primer ejemplo continúa basándose en cámaras de alta resolución para secuenciar, el segundo se basa en el pH.

-Pacific Biociences: Su secuenciador (Pacbio RS II) se basa en el uso de celdas SRMT de un solo que contienen usa polimerasa anclada en su interior. Se le añaden los nucleótidos marcados fluorescentemente que serán excitados por la luz del fondo de la celda  para emitir su luz cuando la polimerasa los atrape. La lectura de una celda puede variar de 30 a 240 min, llegado a tener lecturas de hasta 40000 pares de bases(PACIFIC BIOSCIENCE, 2014).

-Life technologies: Tiene secuenciadores Ion Torrent (como el Ion Personal Genome Machine® (PGM™) System). Tiene un paso de amplificación similar a 454. Estos secuenciadores tienen la particularidad de que no necesitan cámaras puesto que se basan en que al incorporarse un nucleótido en la nueva hebra de DNA, se libera un H+ de modo que puede detectarse con su pH-metro. De modo que se van inyectando los nucleótidos y según haya variación en el pH o no, así como la magnitud de la variación se sabe cuántos se han incorporado.

La longitud de las lecturas varía de 35 a 400 bases (de media 200) tardando menos de tres horas. Pero la capacidad del secuenciador es mayor, puesto que cada secuenciador lee un Chip (el soporte donde se encuentran los fragmentos de DNA) entero a la vez y este contiene más de un millón de pocillos, con lo cual en las tres horas se pueden leer perfectamente más de 200 millones de pares de bases(LIFE TECHNOLOGIES, 2015).

En la figura 4 puede verse el descenso general de los precios hasta este año. Se puede apreciar que se está estancando el precio e incluso ha incrementado ligeramente

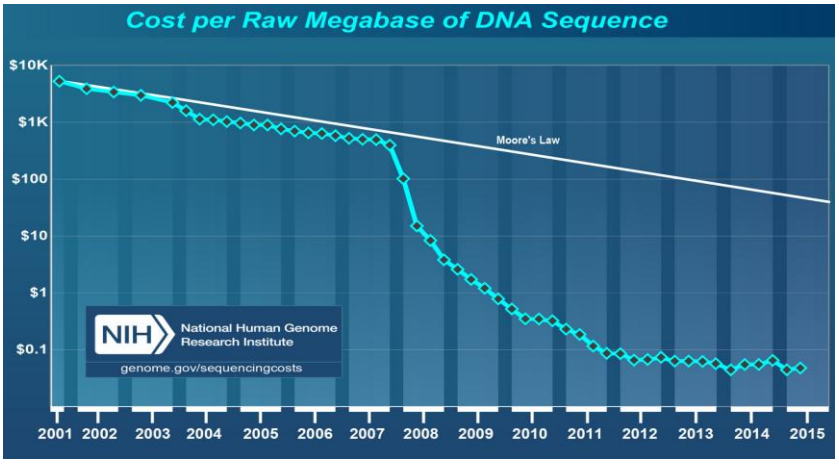


Figura 4: Evolución del precio de secuenciación según www.genome.gov

## 1.2 Sistemas de genotipado masivo

El genotipado de un individuo consiste en descubrir que variaciones presenta en su genoma respecto a otros individuos de la misma especie. Una forma de genotipar es hacer uso de las tecnologías de secuenciación anteriormente mencionadas para secuenciar ya sea todo o solo una parte del genoma de cada uno de los individuos a estudiar y comparar las secuencias a fin de hallar diferencias entre ellas. Otra opción es usar Arrays de genotipado o GBS.

### 1.2.1 Arrays de genotipado

Dado que no todas las especies han sido secuenciadas no hay Arrays comerciales para muchos organismos. Sin embargo Illumina ofrece también la posibilidad de pedir un Array personalizado con los oligonucleótidos que crea convenientes el investigador.

Existen distintos métodos de genotipado por arrays. Uno de ellos es la tecnología de microarrays BeadArray que no es más que un soporte de fibra óptica o sílice cubierto por microperlas de sílice de 3 micrómetros. Cada microperla tiene cientos de copias de un determinado oligonucleótido. A este array se le añade el DNA del cual se buscan los SNP, previamente amplificado, de modo que los fragmentos de DNA quedarán unidos a los oligos si poseen la región complementaria. Los oligonucleótidos tienen una longitud de 50 pares de bases y están diseñados de modo que la base número 51 sea el SNP, con lo cual al introducir nucleótidos marcados fluorescentemente el color que tenga indicará que base hay en nuestro DNA para cada SNP (ILLUMINA, 2015).

### 1.2.2 GBS

Es una técnica nacida tras la aparición de las tecnologías de secuenciación masiva. Su funcionamiento puede resumirse como muestra la figura 5 en unos pocos pasos. En primer lugar cortar el DNA con un enzima de restricción que corte frecuentemente. A continuación ligar unos adaptadores con 'código de barra' (una secuencia de 4 a 8 nucleótidos), seguido por la amplificación de esos fragmentos usando primers complementarios a los adaptadores y por último, la secuenciación de sus extremos. Es posible encontrar 25000 SNP en un solo experimento (Davey, et al, 2011; CORNELL, 2013).

Debido a que las distintas muestras serán de la misma especie el enzima cortará en prácticamente casi todas las muestras en las mismas zonas y se obtendrán los mismos fragmentos (salvo pequeñas variaciones debidas a que el enzima corte o no algún punto concreto por azar o que el SNP haya afectado a la zona de corte).Esta técnica no está limitada a usar un único enzima, se pueden usar varios (Donato, et al, 2013)

Como ya no secuenciamos todo el genoma, si no fragmentos de este reducimos enormemente la cantidad que hay que secuenciar.

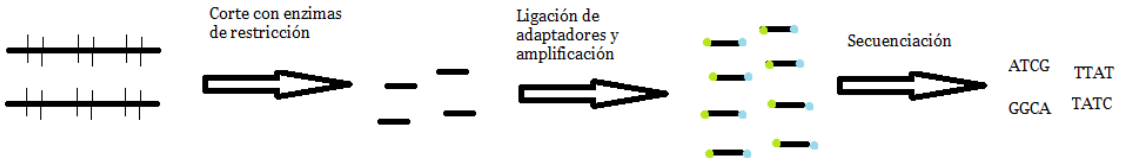


Figura 5: Esquema general del GBS

## 1.3. Ficheros VCF

Los VCF (Variant Call Format) contienen información de la variabilidad de las muestras analizadas, respecto a un genotipo de referencia.

El formato de archivo VCF surgió de la necesidad de disponer de un formato estándar donde guardar los datos de genotipado de individuos y poblaciones. Este formato ha sido uno de los usados como estándar en proyectos como el de 1000 Genomas.

En este trabajo se van a trabajar con ficheros VCF, los cuales no tienen la secuencia entera sino SNP, en decir pequeñas variaciones en el genoma entre muestras.

Los ficheros VCF contienen para cada muestra y para cada SNP información sobre el genotipo, como si es el alelo de referencia o uno alternativo o la calidad. También contiene información general para cada SNP, por ejemplo, su localización (tanto cromosoma como posición en él) o qué bases son las de referencia y las alternativas. En la figura 6, puede verse un ejemplo de las primeras líneas del fichero donde el encabezado (marcado con ## al inicio de la línea) contiene información sobre el fichero. Luego hay una línea que empieza con un solo # donde se indica que información hay en las primeras columnas (el cromosoma, la posición del SNP, el genoma de referencia para dicho SNP, las variantes, la calidad, el filtro, seguido de algunas informaciones). Tras esta hay otra columna, llamada FORMAT la cual puede variar de un VCF a otro. En este caso se indica que genotipo, calidad del genotipo, profundidad de lectura u calidad del haplotipo se encuentran separados por ':'. En este caso solo hay 4 informaciones, pero pueden haber más, no hay ninguna norma sobre cuántas hay que poner. Tras FORMAT aparece el nombre de cada una de las muestras.

Una vez se tiene el VCF, el analista que va a trabajar con ella debe filtrarla y extraer la información que necesite (por ejemplo en un momento dado puede interesarse solo por los genotipos y la calidad, en otro puede estar interesado en la localización de algunos SNP).

```
##fileformat=VCFv4.1
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF    ALT     QUAL FILTER INFO                              FORMAT      NA00001        NA00002        NA00003
20     14370   rs6054257 G      A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2           GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20     17330   .         T      A       3    q10    NS=3;DP=11;AF=0.017               GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:3:5:65,3   0/0:41:3
20     1110696 rs6040355 A      G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2   2/2:35:4
20     1230237 .         T      .       47   PASS   NS=3;DP=13;AA=T                   GT:GQ:DP:HQ 0|0:54:7:56,60 0|0:48:4:51,51 0/0:61:2
20     1234567 microsat1 GTC    G,GTCT  50   PASS   NS=3;DP=9;AA=G                    GT:GQ:DP    0/1:35:4       0/2:17:2       1/1:40:3
```

Figura 6: Ejemplo del formato VCF extraído de 'The Variant Call Format (VCF) Version 4.1 Specification'

## 1.4 Bioinformática

### 1.4.1 La necesidad de la automatización de los análisis

El gran número de secuencias ha convertido la bioinformática en una herramienta fundamental en la genómica. Resulta importante saber programar o disponer de programas ya creados que resuelvan el problema. En estos momentos, con las nuevas tecnologías de secuenciación, resulta imprescindible contar con un bioinformático. El fichero VCF más grande usado en este trabajo tiene 868978 SNP de 348 muestras, trabajar con ello con un excel es imposible simplemente porque un excel no puede abrir un fichero tan grande. Y aunque se pudiese solo hay que pensar en el trabajo y tiempo que le llevaría a una única persona trabajar con tanta información sin saber programar. Solo el hecho de filtrar la información según la calidad de cada SNP implicaría recorrer una a una 868978 filas buscando la casilla con las calidades y eliminando aquellos SNP que no llegasen al mínimo. El filtrado costaría horas de trabajo monótono y posiblemente el investigador a los 50000 SNP ya hubiese empezado a cometer errores como equivocarse con cual era la calidad mínima o saltarse alguna línea. Según fuese avanzando el investigador, los errores serían más frecuentes.

Y todo ese tiempo sería para el fichero anteriormente mencionado, que aunque en este trabajo es el más grande, actualmente no se acerca a los que son verdaderamente grandes. Además, en un par de años será considerado muy pequeño puesto que se prevé que en poco tiempo empiecen a ser frecuentes ficheros de entre 10 y 100 millones de SNP y varios miles de muestras. ¿Cuánta gente y cuánto tiempo tardarían en filtrar un fichero como ese a mano? ¿Cuántos errores cometerían?

### 1.4.2 Python

Python es un lenguaje de alto nivel, esto significa que a la hora de darle las órdenes al ordenador es más parecido a darle órdenes a una persona, a diferencia de lenguajes de bajo nivel en los que te comunicas en el ordenador en un lenguaje más cercano al del ordenador. Otra ventaja es que en Python hay multitud de librerías gratuitas que no son más que fragmentos de programas que puedes usar fácilmente en tu programa de modo que pueden facilitarte mucho el trabajo.

El hecho de usar un lenguaje de alto nivel tiene sus ventajas y sus desventajas. El hecho de ser más fácil de aprender y usar por sus similitudes con el lenguaje humano, aunque parezcan ventajas que solo facilitan la vida a quien no se atreve con lenguajes más difíciles conlleva otras ventajas indirectamente. El hecho de ser más fácil de aprender implica un dominio mayor del lenguaje. Un mayor dominio junto a su facilidad de uso implica poder diseñar mejores programas, ya que igual que una persona trata de transmitir sus ideas a otras personas usando un idioma que domina tendrá más éxito que si trata de realizarlo en una lengua que no domina del todo. Por ello el programador logrará realizar un programa de una complejidad similar en menos tiempo si usa un lenguaje de alto nivel.

Usar un lenguaje de bajo nivel, también tiene ventajas como mayor control del uso de la memoria o poder dar órdenes más precisas. Aunque dominarlo lleva mucho más trabajo se puede conseguir hacer aplicaciones más eficientes.

Durante el proceso de creación de un programa hay que tener en cuenta dos factores: el tiempo de creación del programa y la eficiencia del programa. En el caso de este

proyecto ya había software disponible, pero este estaba diseñado para trabajar con una cantidad inferior de SNPs de la cantidad de SNPs con que se trabaja hoy en día. Por ejemplo, el programa filtraba en 4 o 5 minutos 3000 SNP y una docena de muestras

Dado el avance en las tecnologías de genotipado se necesitan métodos más rápidos de análisis.

En este proyecto es la velocidad el problema que se busca solucionar, pues estamos en el punto de tardar varias horas para filtrar VCF que no sean de los más pequeños. Luego acceder a la información para hacer los análisis también lleva horas de cómputo por parte del ordenador.

## 1.5. HDF5

Una forma de guardar y acceder a la información de forma eficiente es usar el formato HDF5.

El formato HDF5 es la última versión del formato ficheros HDF (Hierarchical Data Format) que fue creado a finales de los años 80 y elegido por la NASA en los 90 entre 15 formatos distintos. Los ficheros de este tipo almacenan la información (usualmente matrices de números, pero también cadenas de texto) dentro de datasets organizados de forma jerárquica en groups. Para simplificar las cosas se puede hacer una analogía con carpetas y ficheros.

Los groups serían como las carpetas y los datasets como un documento cualquiera (un excel por ejemplo). De modo que es posible crear tantos grupos o datasets dentro de otro grupo al igual que se pueden crear tantas carpetas o documentos dentro de una carpeta. Sin embargo dentro de un documento se puede guardar la información que quieras pero no puedes crear otros documentos o carpetas dentro de él.

Por tanto se puede guardar la información en el orden que queramos teniendo en cuenta esto.

Por otro lado la información que se puede guardar es en principio cualquiera que se pueda escribir. Sin embargo no toda es igual de fácil de guardar. La información que normalmente se guarda dentro de un dataset son matrices de números. Estás matrices de datos pueden crearse usando librerías como NumPy (Numeric Python). En NumPy a las matrices se les llama arrays.

Aunque lo normal sean matrices de números también se pueden guardar cadenas de texto.

A los ficheros creados bajo el formato HDF5, se les denomina ficheros HDF5. HDF5 también es el nombre que recibe una librería. La librería HDF5 es software libre. El software libre se caracteriza porque el usuario que lo adquiere puede ejecutar el código, copiarlo, estudiarlo, modificarlo y distribuirlo modificado o sin modificar, libremente. Además este tipo de software suele estar disponible gratuitamente (como en el caso de esta librería).

Esta librería está mantenida por el HDF Group, sin ánimo de lucro. Está escrita principalmente en C y algunas partes en C++ y Java. PyTables y h5py son las interfaces más conocidas de Python, diseñadas para usar la librería en C del HDF Group. Para este proyecto se usó h5py.

# 2. OBJETIVOS

Al inicio de este trabajo existía el problema de la lentitud al trabajar con los ficheros VCF a la hora de extraer y analizar los datos de genotipado. Una posible solución era utilizar la librería VCFNP. Esta librería permite guardar los datos contenidos en un fichero VCF en un fichero HDF5.Como alternativa se diseñó otra librería que fuese capaz de guardar los datos de genotipado de cualquier VCF, independientemente de su tamaño, en un fichero HDF5. Por otro, lado para poder analizar los datos de genotipado contenidos en un HDF5 era necesario desarrollar herramientas para filtrar las grandes matrices de genotipado así como para realizar los cálculos pertinentes para su análisis.

Por ello los objetivos de este trabajo son:

-Conseguir analizar la información que pueda contener un VCF de forma más rápida, para lo cual se:

-Buscarán herramientas disponibles o se crearán herramientas para:

Transformar ficheros VCF en HDF5

Filtrar ficheros HDF5 de SNPs

Hacer análisis con los genotipos contenidos en los ficheros HDF5

-Evaluará la eficiencia de las herramientas HDF5 orientadas a datos de genotipado

# 3. MATERIALES Y MÉTODOS

## 3.1 Equipo informático

Portátil con las siguientes características:

-SO: Ubuntu 14.04 LTS

-Tipo de SO: 64 bits

-Memoria: 2,9 GiB

-Procesador: Intel Core 2 Duo CPU T7300 @ 2.00GHz x 2

-Gráficos: Intel 965GM

-Disco: 475,2 GB

## 3.2 Ficheros VCF de prueba

Para realizar las pruebas de las distintas herramientas, tanto preexistentes como creadas durante el proyecto se usaron 3 ficheros VCF.

-Ril.vcf: 153 muestras y 943 SNPs

-tomato.apeki_gbs.vcf: 12 muestras y 41682 SNPs

-tomatos.vcf: 348 muestras y 868978 SNPs

## 3.3 Librería VCFNP

La librería VCFNP permite guardar datos de los VCF en HDF5. Se puede encontrar en https://github.com/alimanfoo/vcfnp. Esta librería hace uso de otras dos librerías (NumPy y Cython), con lo cual hay que instalarlas para poder usarla. Esta librería carece de documentación, aunque posee algunas líneas con ayudas en los scripts (ficheros de texto con código) que facilitan entender el código.

Para este proyecto se usaron los scripts llamados vcf2npy y vcfnpy2hdf5 de esta librería. El primero obtiene a partir de parte de un VCF un fichero NPY, que no es más que un conjunto de matrices de NumPy que contiene la parte del VCF que se ha pedido. El segundo transforma el fichero NPY resultante del primer script en un fichero HDF5.

## 3.4 Sistema de control de versiones, Git

El control de versiones no es más que una forma de gestionar los cambios que se van realizando durante un proyecto, de modo que se pueda acceder al estado que se encontraba el proyecto en un determinado momento.

Existe una versión por cada miembro del proyecto, más una versión extra en la que se reciben los cambios que desean enviar los miembros del proyecto y desde donde se actualizan sus versiones cuando así lo quieran. Cada miembro puede hacer los cambios que quiera en los archivos, puede eliminarlos o crear nuevos sin afectar a los demás siempre y cuando no los comparta.

Para registrar una modificación hay que usar el comando git commit seguido del nombre del archivo y -m junto a un comentario que se desee realizar referente a la modificación del archivo.

Cuando el miembro del grupo ha acabado su parte y quiere compartirla con los demás debe usar el siguiente comando: git push. De esta forma sus cambios actualizan esa versión máster que registra todas estas actualizaciones. Ahora si los demás miembros del grupo desean recibir estos cambios deben usar git pull.

Cuando se usa tanto push como pull hay que tener cuidado, puesto que *si* varios miembros del grupo han realizado cambios sobre un mismo archivo puede haber un conflicto y no se podrá actualizar este archivo en concreto automáticamente por lo que se deberá hacer a mano.

Durante el proyecto o al finalizarlo puede ser interesante acceder a un punto concreto del desarrollo. Para ello se usa git log. Al usarlo aparece un registro con todos los cambios realizados. Aparece un código de identificación (como por ejemplo f3b4c89fd2d0539a3d8a5e20b30ff34a3b0da58c), el autor del commit, la fecha en la que se realizó y el comentario que se hizo. Si se quiere acceder a una versión en concreto solo hay que usar git fetch seguido del código que aparece en el log para el commit deseado.

En este proyecto se usó esto una vez finalizado para comprobar si en algún cambio había provocado una pérdida de compresión o de velocidad, a fin de ver si era posible mejorar la versión final.

## 3.5 Librería allel

La librería allel dispone de diversas funciones que trabajan con los datos contenidos en un VCF, en especial los de genotipo. Algunas de estas funciones sirven por ejemplo para el cálculo del ACP o del equilibrio de ligamiento

El autor de esta librería es el mismo que el de la librería VCFNP. Por ello está diseñada para partir de la información guardada en un HDF5, aunque también serviría un array de NumPy.
En este caso si hay documentación que se puede leer en http://scikit-allel.readthedocs.org/en/latest/index.html
Esta librería necesita para funcionar tener instaladas algunas librerías: NumPy, scipy, matplotlib, pandas, scikit-learn, h5py, numexpr, bcolz y petl.

La mayor parte de las funciones requieren de los datos genotype de sus HDF5, que no es más que la parte GT de un VCF pasada de este formato ‘0/0’ o este ‘0|0’ a este [0, 0] y guardada en un array de NumPy.

### 3.5.1 GenotypeArray

El primer paso para trabajar con los genotipos una vez guardados en un fichero HDF5 es obtener el GenotypeArray. Esto es transformar el array de NumPy cargado desde el fichero HDF5 en un GenotypeArray (la información de los genotipos sigue siendo la misma y tiene la misma estructura, solo que ahora se le considera un objeto), mediante el uso de allel.model.GenotypeArray()

A modo de ejemplo, si se tiene el GT dentro de ril.HDF5 simplemente habría que abrir el fichero en modo de lectura (‘r’) y guardar en una variable GT del siguiente modo:

```
f = h5py.File('ril.HDF5', 'r')
genotypes =  f['calldata']['genotypes']
```

Entre corchetes se le da el nombre de los grupos  hasta llegar al dataset (solo hay un grupo llamado ‘calldata’)  y le damos entonces el nombre del dataset(‘genotypes’).

Ahora solo hay que guardar en otra variable el GenotypeArray:

```
g = allel.model.GenotypeArray(genotypes)
```

### 3.5.2 ACP

El cálculo del ACP es uno de los cálculos disponibles en la librería allel. Durante el proyecto se probaron otras herramientas, tales como la heterocigosidad esperada o el desequilibrio de ligamiento, sin embargo solo se va a mostrar el cálculo del ACP. Se ha elegido el ACP como ejemplo de potencial de la librería allel debido a que es de los más costosos en tiempo. Otra razón de elegir el ACP como ejemplo se debe a que se presta más a ser mostrado en una animación.

Calcular a partir de un GenotypeArray  los puntos del ACP requiere un paso previo:
`gn = g.to_n_alt()`
Esto lo que hace es asignar un número para cada genotipo y muestra según los alelos que posea, la asignación es la siguiente en el caso de diploides:

-A los homocigotos para el alelo de referencia se les asigna el  0.
-A los heterocigotos se les asigna el 1.
-A los que poseen dos alelos alternativos se les asigna el 2.

Hay que tener en cuenta que esta codificación confunde los distintos posibles alelos alternativos.Por ejemplo dos alelos alternativos tanto los individuos que tengan para un SNP [0,1] o [0,2] ambos van a ser considerados 1, del mismo modo aquellos que sean tanto [1,1], [2,2] o [1,2] van a ser considerados como 2.

El cálculo de los componentes principales puede requerir de un filtrado previo de gn. Esto se debe a que si algún SNP no tiene datos o es monofórmico para el alelo 2, el cálculo falla.
Una vez hecho el filtrado, para realizar el cálculo a partir de gn basta con hacer lo siguiente:

```
ACP = allel.stats.decomposition.ACP(gn)
```

Si se hace de este modo hay que tener en cuenta que la función toma una serie de parámetros por defecto:

-n_components: Por defecto es 10

-copy: True

-scaler: 'patterson'

-ploidy: 2

Si se quiere cambiar cualquiera de los parámetros por defecto basta con incluirlos tras gn, separados con comas y con el valor deseado. A modo de ejemplo, si solo se quisieran obtener 5 componentes y el organismo fuera triploide haríamos lo siguiente:

```
ACP=allel.stats.decomposition.ACP(gn, n_components=5, ploidy=3)
```

En líneas generales el funcionamiento del cálculo del ACP es el siguiente:

-Transforma la matriz gn en un allel.stats.preprocessing.PattersonScaler object

-Transforma el objeto en una matriz y la transpone

-Obtiene el número de muestras en la matriz y en número de SNP (por el número de filas y el de columnas)

-Obtiene u, s y v usando la librería scipy.linalg.

-u es la matriz unitaria que posee los vectores singulares de la izquierda como columnas. Mismas dimensiones que la matriz transpuesta.

-s son los valores singulares ordenados en orden no ascendente. Su dimensión en el número de SNP

-v es la matriz unitaria que posee los vectores singulares como filas. Sus dimensiones son número de SNP x número de SNP.

-Calcula la varianza explicada como $s^2$/nº de muestras

-Calcula el ratio de varianza explicada dividiendo la varianza explicada entre la suma de las varianzas explicadas.

-Hace *slices* de v, de la varianza explicada y del ratio de varianza explicada, cogiendo hasta el valor del número de componentes dado.

-Luego hace lo mismo con u y con s y los multiplica. El resultado lo transforma en un array y lo entrega como las coordenadas del ACP.

## 3.6 Medidas de tiempo

Para realizar todas las medidas de tiempo se usó el comando 'time' delante del nombre del script a ejecutar. Con este comando cuando la ejecución del script ha terminado se imprime en pantalla el tiempo que ha tardado en realizar la tarea.

Las medidas de tiempo fueron tomadas usando el ordenador exclusivamente para ejecutar los scripts (es decir no había ningún programa ni ninguna ventana más abiertas a excepción del script en cuestión y el terminal)

### 3.6.1 Creación de los archivos HDF5

#### 3.6.1.1 VCFNP

En el caso de la librería VCFNP se midió el coste de tiempo de VCF2NPY 5 veces para cada combinación de archivo output (calldata, calldata_2d y variants) y cada VCF. En el caso del VCF de tomates el tiempo de calldata_2d y calldata fue medido usando la opción de exclude_fields de modo que solo se incluyese GT (los genotipos).

El tiempo de VCFNPY2HDF5 se midió usando los archivos obtenidos con el script VCF2NPY. 5 medidas de tiempo por cada archivo.

#### 3.6.1.2 Variation 5

Se realizaron 5 medidas de tiempo en todos los casos.

En el caso de ril se hicieron las medidas de tiempo para SNPS_PER_CHUNK con valor de 200 y 1000 tanto para shuffle y fletcher32 'False' como 'True'.

Para apeki se midió el tiempo dándole a la variable SNPS_PER_CHUNK el valor de 200, 10000 y 25000 en el caso de ser las variables shuffle y fletcher32 'True' y 200 y 25000 en caso de ser 'False'. Dos de las medidas de SNPS_PER_CHUNK=200 y shuffle y fletcher32 'False' se tomaron a la vez.

Con tomatos se tomaron las medidas usando SNPS_PER_CHUNK=200. Para dos de los bloques de 5 medidas de tiempo se utilizó kept_fields=['GT'] de modo que solo se incluyesen los genotipos. Un conjunto de medidas de tiempo se hizo con shuffle y fletcher32 'True' y el otro 'False'. Por último se hicieron las 5 medidas para ver cuánto tardaba en guardar todo el fichero con shuffle y fletcher32 'True', salvo la primera medida, en este caso las medidas fueron con el navegador abierto (es decir se mantuvieron la página de google Drive abierta simulando un incremento de recursos del sistema al realizar otra tarea con el ordenador)

### 3.6.2 Tiempo de filtrado

Para estimar el tiempo de filtrado se probó por un lado para cada fichero, lo que costaba por un lado cada uno de los dos primeros filtros (estos filtros ya no están en la versión final del script de filtrado).

Antes de eliminar estos dos filtros, se midió el tiempo del filtro af que usaba np.vstack, el mismo bloqueando la parte de código donde se montaba el array de nuevo, y el filtro de gn.

Por otro se midió el tiempo de filtros finales de forma aditiva. Primero se aplicó solo el filtro filter_snps_by_maf. Tras tomar las medidas de tiempo se procedió a tomar de nuevo las medidas esta vez aplicando dos filtros seguidos (filter_snps_by_maf junto al filter_snps_by_missing_calls). Al finalizar se volvieron a tomar las medidas de tiempo aplicando tres filtros. Por último se volvió a medir el tiempo aplicando los cuatro filtros. Se ejecuto siempre solo el código imprescindible para que se realizarán los filtros.

En todos los casos fueron 5 medidas de tiempo para los datos de genotipo de ril, otras 5 para los de apeki y otras 5 para los de tomatos.

### 3.6.3 Estimación del tiempo del ACP

En el caso del ACP se utilizó el tom_call2d.HDF5 y se probó la velocidad con distintas cantidades de SNPs y distinto número de muestras .

De nuevo se tomaron 5 medidas en cada caso.

## 3.7 Matplotlib

Matplotlib es una librería de Python, que provee herramientas que permiten representar nuestros datos con distintos tipos de gráficos bidimensionales o tridimensionales. Haciendo uso de estas herramientas es posible crear un pequeño script que nos permita representar las proyecciones obtenidas en el ACP. La opción que se eligió fue una representación tridimensional de estas proyecciones. Se crearon distintas imágenes tridimensionales de cada resultado para obtener una serie de fotogramas que, una vez unidos, creasen una rotación alrededor del eje z.

## 3.8 Avconv

Avconv, es una de las utilidades de la librería Libav, la cual posee multitud de herramientas disponibles para conversiones de formatos de video, montajes, extraer o añadir sonido.

En este caso se usó la herramienta de montaje de video a partir de un conjunto de imágenes a fin de obtener una pequeña animación a partir de las imágenes generadas con Matplotlib.

Para obtener el vídeo se usó el comando:

```
avconv -f image2 -i image%04d.png video.avi
```

### 3.9 KMeans

A fin de comprobar el desempeño de las matrices de genotipado con otra herramienta, se añadió una pequeña función a filtros_ACP.py que permitía agrupar en clusters. Para el cálculo se usa la clase Kmeans de la librería scikit-learn.

# 4. RESULTADOS Y DISCUSIÓN

## 4.1 HDF5

Para guardar los genotipos en un fichero HDF5 primero se debe crear el fichero HDF5. Por ejemplo llamado prueba.HDF5, en modo de escritura ('w'):

```
fhand = h5py.File('prueba.HDF5', 'w')
```

Para la creación de un dataset (los datasets es el nombre que reciben en los HDF5 las matrices de datos tanto numéricos como en forma de texto) debe dársele un nombre y los datos que se quieren guardar en él o sus dimensiones. El dataset puede ser creado dentro de un grupo o directamente en el directorio raíz del HDF5. En este último caso la creación se haría directamente desde el fichero abierto.

```
Dtset = fhand.create_dataset('directo', data=nums, dtype=dtype)
```

Para crear un grupo hay que especificar el nombre del grupo y dónde se desea crear.

```
Grp = fhand.create_group('primero')
```

Si se quiere crear otro grupo dentro de ese grupo:

```
Grupo_2 = Grp.create_group('segundo')
```

De forma similar se haría si se desea crear el dataset anterior dentro de un grupo:

```
Dtset = Grupo_2. create_dataset('directo', data=nums, dtype=dtype)
```

Para acceder a un grupo o dataset se usan los nombres de groups y datasets del mismo modo que en un diccionario de Python. Por ejemplo, si se quisiera acceder a la información de un dataset se haría del siguiente modo.

Primero se abriría el HDF5 en modo de lectura ('r'):

```
fhand = h5py.File('prueba-HDF5', 'r')
```

Y tras esto solo se usarían los nombres de uno de estos modos:

```
Datos = Fhand['primero']['segundo']['directo']
```

O alternativamente:

```
Datos = Fhand['primero/segundo/directo']
```

Los datasets pueden ser indexados, de modo análogo a como se indexan los arrays de numpy:

```
Diez_datos = Datos[:9]
```

Si el array tuviese varias dimensiones y se desease seleccionar, por ejemplo, las tres primeras columnas:

```
Datos = Fhand['primero']['segundo']['directo'][:,:2]
```

Por ejemplo si en el dataset 'directo' hubiese un array de NumPy con 50 SNP (primera dimensión), de 25 muestras (segunda dimensión) de un organismo diploide (tercera dimensión) y por cualquier motivo deseamos únicamente coger la información de los SNP del 20 al 27, de las muestras 3 a 18 y solo del primer alelo, se haría lo siguiente:

```
seleccion = Fhand['primero']['segundo']['directo'][19:26, 2:17, :1]
```

Los ficheros HDF5 pueden ser explorados con el programa ViTables. Tal y como se ve en la imagen se puede tener más de un dataset abierto simultáneamente.

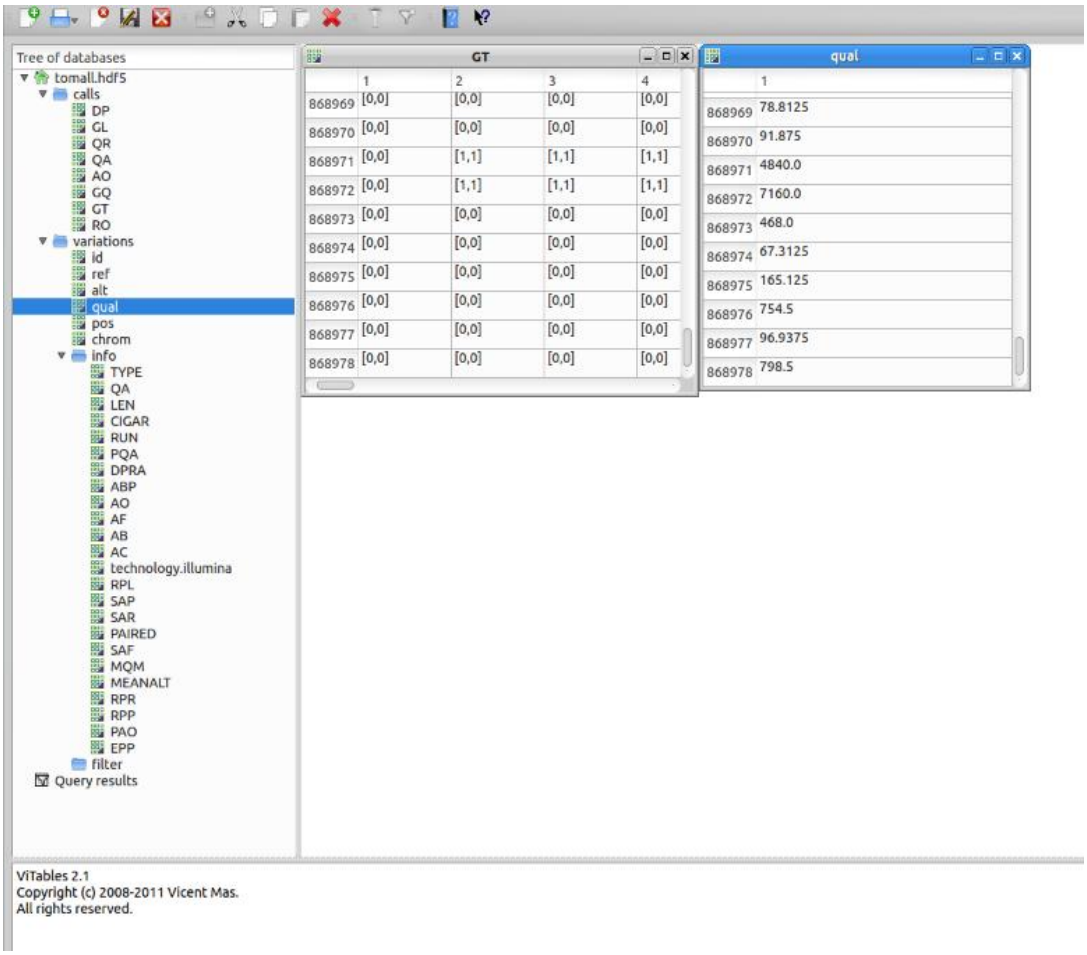


Figura 7: Ejemplo de la Interfaz de ViTables

## 4.2 Librería VCFNP

### 4.2.1 VCF2NPY

El script VCF2NPY de la librería VCFNP extrae la información de un fichero VCF y crea un fichero con arrays de NumPy. Cuenta con diversas opciones para elegir una parte del VCF:

Se pueden obtener tres tipos de ficheros diferentes:

-Variants: Contiene la información del fichero VCf no relacionada con los genotipos de las muestras/individuos. Por ejemplo, aquí se encontraría información como el genotipo de referencia y los alternativos, el cromosoma en el que se encuentra y la posición en el.

-Calldata: Contiene el genotipo para cada muestra, además de la calidad del genotipo o cualquier otra información asociada. Por ejemplo, aquí se encuentra información sobre si el SNP es como el de referencia o alguno de los alternativos o la calidad de la lectura.

-Calldata_2d: Contiene la misma información que calldata, pero en lugar de un solo dataset, tiene un dataset por cada tipo de información, por ejemplo, el genotipo y la calidad del mismo se encontrarían en dos datasets diferentes.

Además, el script Vcf2npy posee para seleccionar secciones del fichero VCF:

-Exclude_fields: Se le puede dar una lista con el nombre de los camposdel fichero VCF para que no sean incluidos en el fichero NPY.

-chromosome: El nombre del cromosoma del que se quiera extraer la información, si se quiere todo se usa None (por defecto)

-task_size: El tamaño en pares de bases de la región de la que se quiere extraer la información (por defecto toda con None)

-fasta: Hay que dar el nombre del archivo fasta que contenga el genoma de referencia. Solo en caso de usar task_size.

En este proyecto se crearon los tres tipos de archivos output diferentes a fin de ver su contenido tanto en el formato NPY, como en el HDF5 obtenidos con ellos.

De entre las opciones extra la única que se empleó fue la de exclude_fields por motivos de capacidad del ordenador tal y como se señalará en resultados.

### 4.2.2. VCFNPY2HDF5

El segundo script parte del fichero NPY para escribir la información en un HDF5 según esté ordenada en este.

Este script cuenta con algunas opciones para la creación del HDF5:

-Compression: Es el string (un string es una cadena de texto) indicando la compresión a aplicar. Si no se quiere comprimir se le puede dar None. El filtro por defecto es ‘gzip’, considerándose el mejor por tener una buena compresión, con una velocidad moderada y pudiéndose elegir el valor de compresión entre 0 y 9 (en compression_opts). En este script el nivel de compresión elegido es 3.

-shuffle: Puede ser False o True (por defecto False). Si se usa, reordena los bits de los fragmentos y puede mejorar el ratio de compresión.

-fletcher32: True o False (por defecto False). Se usa para detectar corrupción de datos.

### 4.2.3 Limitaciones de la librería VCFNP

La librería VCF guarda la información de los ficheros VCF en distintos tipos de ficheros HDF5 (variants y el calldata o el calldata_2d). Este hecho permite seleccionar que información guardar, pero a la vez implica que de querer conservar toda la información del fichero VCF se obtendrá más de un fichero HDF5. Esto implica que o bien se debe crear un script que guarde la información de un conjunto de ficheros HDF5 en uno solo, o conservar la información del fichero VCF repartida entre distintos ficheros HDF5 con los posibles errores que puede conllevar (por ejemplo filtrar un fichero HDF5, pero no los otros).

El segundo problema, y el verdaderamente importante, apareció durante el paso de VCF a NPY del fichero de tomatos.vcf. Para el ril.vcf y para el tomato.apeki_gbs.vcf no hubo problemas, tampoco para generar el variants de tomatos.vcf, pero sí para generar el calldata y el calldata_2d. En estos dos últimos casos, tras aproximadamente media hora de estar llenándose la matriz de datos, el script se detuvo por falta de memoria RAM habiendo alcanzado casi 200000 SNP. Por tanto a partir de este fichero, que aun siendo bastante grande no es de los más grandes existentes, no es posible obtener un calldata o un calldata_2d en un ordenador modesto (3GB de RAM). A fin de poder usar los datos de los genotipos de este archivo, se usó el campo de exclude_fields, dejando únicamente los genotipos. Este segundo problema agrava el primer problema, pues implica que si se desea conservar la información de un VCF grande en NPY o en HDF5, usando un ordenador que no tenga mucha RAM, habrá que hacerlo en muchos ficheros o crear un script que vaya fusionándolos.

A parte, si se desea conservar la información en estos formatos, habría que guardar el encabezado del fichero VCF a parte, puesto que no se incluye en ningún fichero HDF5.

### 4.2.4 Velocidad y tamaño de los ficheros

Para ver la eficiencia de la librería VCFNP se midió el tiempo y el tamaño de los archivos resultantes. En el anexo tiempo_npy_hdf5, se recogen los datos de tiempo obtenidos en la conversión en NPY y HDF5 para cada caso. En este anexo puede verse como la primera medida de tiempo en el paso de VCF a NPY es superior a las siguientes. En el caso del paso de NPY a HDF5 se observa que las cinco medidas medidas de tiempo son similares en todos los casos (salvo para variants de ril). Esta

diferencia de comportamiento puede deberse a que en el caso de la creación del fichero NPY, si este ya existe, es cargado de la caché. Sin embargo, si el HDF5 ya existe, se vuelve a crear como si no existiese.

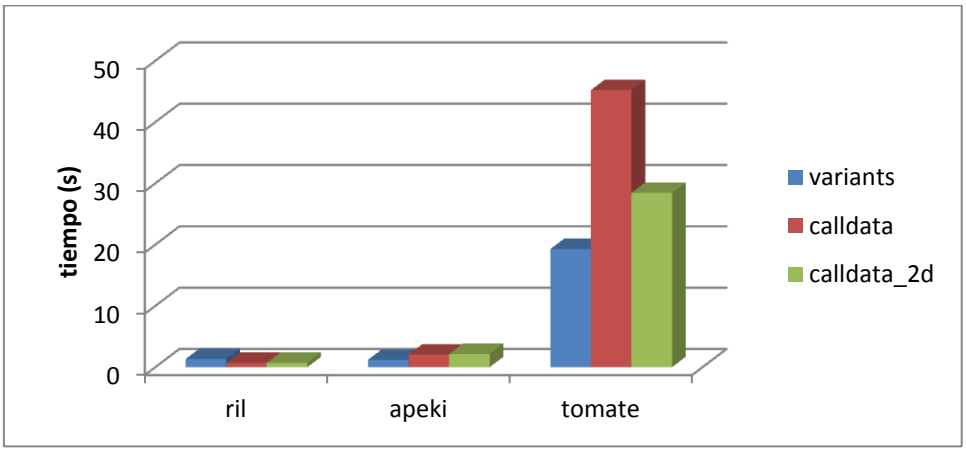


Figura 8 : Gráfica del coste de tiempo en transformar distintos archivos VCF en NPY

El paso de NPY a HDF5 es más rápido que el de VCF a NPY. Tal y como puede verse en la figura 9.

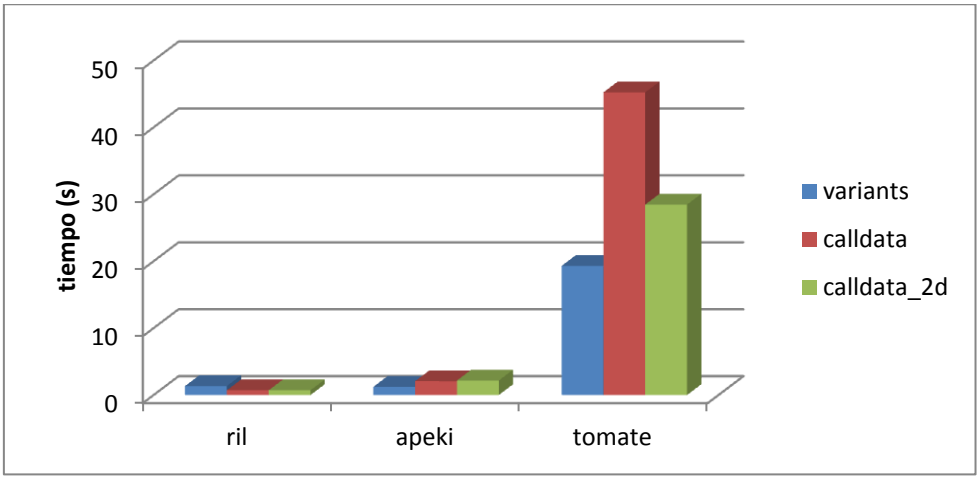


Figura 9: Gráfica del coste de tiempo en transformar distintos archivos NPY en HDF5

Por otro lado, al comparar espacio en disco ocupado por los ficheros obtenidos respecto a los ficheros VCF originales, se puede ver lo siguiente para los ficheros NPY:

-En el caso de ril.vcf, el tamaño de variants junto al de calldata_2d es mayor al del VCF descomprimido.
-Si variants y calldata_2d, provenientes de ril.vcf, son comprimidos su tamaño es similar al VCF comprimido.
-Los ficheros NPY creados a partir de apeki.vcf tienen un comportamiento similar al de los dos puntos anteriores.
-Los ficheros calldata_2d y variants creados a partir de tomatoes.vcf ocupan menos que el VCF comprimido.

-En el caso de los ficheros calldata_2d y variants comprimidos provenientes de tomatoes.vcf, ocupan aproximadamente 16 veces menos que el VCF comprimido.

Una vez comparado el tamaño de los NPY obtenidos, respecto a los VCF originales, se procedió a hacer la comparación de los HDF5 con los VCF:

-Los dos HDF5 de ril tienen un tamaño mayor al VCF comprimido.
-Los dos HDF5 de ril tienen un tamaño menor al VCF descomprimido.
-Los dos HDF5 de apeki se comportan como los HDF5 de ril en cuanto a compresión.
-Los HDF5 de tomatoes ocupan menos que el VCF descomprimido.
-Los HDF5 de tomatoes comprimidos ocupan en torno a 10 veces menos que el VCF descomprimido

Hay que tener en cuenta que la compresión no afectó del mismo modo a los ficheros NPY y a los HDF5. Si se compara el ahorro de comprimir los 3 ficheros en cada caso se obtiene que mientras con los NPY se reduce el tamaño en torno a 14, 4 y 10 veces en los casos de ril, apeki y tomate respectivamente, en el caso de los HDF5 la reducción apenas existe siendo para los mismos casos de 1.21, 1.03 y 1.03, que haya tan poca reducción del tamaño en el caso del HDF5 es lógico pues durante su creación ya se comprimen los datos.

Finalmente si se comparan los tamaño finales se obtiene que si no se comprimen los ficheros, la opción óptima de almacenamiento son los HDF5 ya que ocupan menos espacio (entre 4 y 5 veces menos en el caso del fichero tomate.vcf) y nos ahorramos unos segundos al no tener que transformar el NPY en HDF5.

Sin embargo, si se comprimen los ficheros NPY ocupan menos (entorno a 1,6 veces en el caso de los tomatos.vcf, diferencia más notable en ficheros más pequeños donde está en torno a 3 veces en el caso de los ril.vcf). Esto supone un dilema a la hora de almacenarlos, puesto que en ambos casos se va a poder acceder a la información muchísimo más rápido y va a ocupar bastante menos que el VCF. Llegados a este punto la mejor opción depende de cada usuario. Una buena conclusión sería que si se va a tener una gran cantidad de información guardada en VCFs, almacenarlos en NPY comprimidos va a suponer ahorrar disco duro. Además, si una vez almacenados raramente vas a usar alguno de ellos, los NPY comprimidos son la mejor opción. En cambio si se prevé que va a sobrar espacio en el disco duro guardándolos como HDF5, lo mejor sería guardarlos como HDF5 descomprimidos, ahorrando unos segundos en descomprimir y pasar de NPY a HDF5. De este modo se ahorra algo de tiempo y la necesidad de usar el script para transformarlos en el momento de trabajar con ellos en el futuro.

A pesar de las ventajas de tiempo y espacio en cualquiera de los dos casos respecto a conservarlo como VCF, hay que remarcar dos desventajas. La primera es que la RAM supone una limitación puesto que trabajar con ficheros como el tomato.vcf o de más tamaño resulta imposible para un ordenador normal, siendo necesario dividirlo en una mayor cantidad de HDF5 con cada una de las partes. Y para ficheros más grandes posiblemente sea imposible obtener por separado cada campo del HDF5 puesto que la RAM no podrá ni siquiera cargar el array de NumPy con los genotipos. La segunda desventaja es que tener los ficheros HDF5 en distintos VCF puede suponer una molestia al trabajar con ellos, pero esta desventaja no supone un problema real puesto que es posible crear un script que fusione los distintos HDF5 en uno solo.

## 4.3 Variation 5

### 4.3.1 Motivo de creación

La librería VCFNP no permite guardar la totalidad de los datos de un fichero VCF en un único HDF5 directamente. Además tiene el problema de depender de la memoria RAM para cargar toda la información de los ficheros a la vez. Depender de la RAM implica que no es posible usar la librería VCFNP con cualquier fichero VCF para obtener un HDF5 (con la información de calldata por ejemplo) de una sola vez. Posiblemente para algunos ficheros VCF muy grandes no sea posible obtener ni siquiera un solo campo (genotypes por ejemplo). Esto hizo necesario desarrollar un programa que si pueda transformar un fichero VCF de cualquier tamaño en un único HDF5 en un único paso. Crear este programa además permite no depender de que se siga manteniendo la librería VCFNP y se le pueden añadir funcionalidades extra que resulten útiles de las cuales carece VCFNP.

### 4.3.2 Control de versiones

Durante el buceo de por las distintas versiones del programa se vieron varios puntos que afectaron significativamente a la velocidad y a la compresión. Uno de ellos permitió pasar de gastar 38MB para guardar solo el calldata de apeki a pesar solo 5,3MB guardando la misma información y además pasar de costar 3 minutos a solo 2.

Otros cambios fueron para arreglar pequeños problemas, como que apareciera una línea extra sin datos al final del HDF5.

A medida que el fichero HDF5 era más completo, los tiempos iban incrementando junto a su tamaño, lo cual en principio indicaría que no es posible mejorar mucho el programa actual sin tener que dedicar un trabajo excesivo. Sin embargo, las optimizaciones de velocidad y compresión en algunas versiones sugieren que se podría buscar para una próxima actualización una ligera mejora.

### 4.3.3 Creación del HDF5

El HDF5 se crea a pedazos, concretamente pedazos de 200 SNP. Al realizarlo de esta manera no se tiene todo el VCF en memoria de golpe, permitiendo que cualquier ordenador sea capaz de pasar de VCF a HDF5 sin importar su memoria RAM.

Por ejemplo, suponiendo que el HDF5 solo va a tener genotipos y calidad. Primero se leería el encabezado y se guardaría en sus datasets correspondientes. Esas líneas se borrarían de la memoria RAM. Tras esto se leerían los primeros 200 SNP, guardándose en primer lugar la información de los genotipos y luego la de calidad. Se borrarían estos 200 SNP de la memoria RAM y se procedería a hacer lo mismo con los 200 SNP siguientes continuamente hasta finalizar.

Variation 5 cuenta con la funcionalidad de exclude_fields del mismo modo que los scripts de la librería VCFNP, solo que aquí se le puede dar cualquier campo sin importar si está o no. Si no está, será ignorado.
Además cuenta con un include_fields, que permite seleccionar los campos que queramos incluir. Añadir esta opción era necesario, puesto que al crearse el HDF5 con

todo el VCF el número de campos total puede ser muy grande. Un elevado número de campos implicaría tener que ecribir más campos en exclude_fields y en este caso siempre se puede dejar sin quitar alguno, con el incremento de coste de tiempo, cuando no se va a necesitar ese campo.

### 4.3.4 Inout.py

A fin de comparar la eficiencia de la librería Variation 5 respecto a la librería VCFNP se midió el tiempo que costaba crear cada HDF5 y su tamaño. En el anexo tiempos_inout pueden verse los resultados obtenidos.

El HDF5 resultante usando este programa es en todos los casos más pequeño que el creado con la librería VCFNP. La librería Variation5 muestra, por tanto, ser más adecuada para conservar el contenido de los VCF en formato HDF5, no solo al obtenerse su contenido en un único HDF5, sino porque también ocupan menos espacio.

Considerando el tamaño de los HDF5 de VCFNP como la suma de calldata y variants se obtiene que los HDF5 con la información de ril, apeki y tomatos tienen un tamaño aproximado de 2Mb, 17MB y 130 MB respectivamente, frente a los 0,5Mb, 3,3-4,3MB y 38Mb. En este último caso hay que recalcar que con variation5 usar kept_fields=[‘GT’] no se conserva únicamente la información de los genotipos sino que también se guarda la información relacionada con variants (es un pequeño bug que se solucionará en una versión futura), con lo cual en las mismas condiciones la diferencia de espacio entre hacerlo con una librería u otra sería superior.

En cuanto a la variable SNPS_PER_CHUNK (los chunks son el tamaño de los pedazos en los que se crea el HDF5, por ejemplo en este caso de 200 en 200 SNPs), ha mostrado tener influencia en la compresión, estando siempre más comprimido el HDF5 cuantos más SNP había en un pedazo. Esto tiene su lógica puesto que si el algoritmo tiene que encontrar la información más común para asignarle valores en su tabla de compresión, va a ser más eficiente cuanta más trozo de la información disponga.

Esta variable también afecta al tiempo, pero solo debido a la cantidad de información que debe haber en memoria. De este modo, en el caso de ril no llega a afectar debido a su bajo número de SNPs. En el de apeki, parece que con pedazos de 25000 empiece a afectar a la velocidad, pero no lo suficiente para determinar que así sea. En cambio, con valores cercanos al total de SNP empieza a tardar mucho más.

Con la información de tomates, esto ocurre con pedazos más pequeños, usando pedazos de 10000 SNP en 40 minutos solo se consigue guardar la información de 10000 genotipos y la información variants de 20000 SNPs. Mientras que solo se necesita media hora para guardar todos los SNP si se hace con 200 SNP por pedazo.

Con esto se puede concluir que dejar por defecto el valor de 200 SNP por pedazo es adecuado, pues incrementarlo no va a conseguir una compresión mucho mayor y no va a mejorar la velocidad, en todo caso la empeorará.

En cuanto al tiempo que tarda esta librería frente a la librería VCFNP, tenemos que ril y apeki tardaron más. Mientras que con VCFNP el paso a NPY y a HDF5 (sumando variants y calldata) costó sobre 6 segundos en el primer caso y 24 segundos en el segundo. Con variants 5 tardó unos 9 segundos en el primer caso y 6 minutos en el

segundo. Con lo cual si se tiene prisa y un fichero muy pequeño usar una u otra librería es indiferente, si se tiene un fichero mayor ya es más interesante usar VCFNP.

Sin embargo, al probar el fichero de tomates, se vió que para los ficheros más grandes de todos la librería variation5 no solo ahorra espacio sino tiempo también, ya que lo que tardaría cerca de una hora con VCFNP tarda tan solo media hora con Variation5. Quizá en Variation 5 el número de muestras condicione significativamente la eficiencia ya que ril con 153 muestras tarda 3s más (aunque relativamente un 50% más), en el caso de apeki solo hay 12 muestras y tarda mucho más (casi 6 minutos más o un 1500%) y en el último caso hay 348 muestras y tarda menos. Si se debe a esto debe haber un número de muestras donde usar Variation 5 o VCFNP suponga el mismo tiempo.

Además con la librería Variation5 se puede guardar el fichero HDF5 entero de una vez, cosa que con VCFNP no se podía. Guardar en un HDF5 toda la información del VCF ya supone una inversión de tiempo más grande puesto que cuesta cerca de tres horas (caso de tomatoes). Unas pequeñas pruebas adicionales con la librería Variation5 para ver lo que costaba realizar a la vez dos HDF5 con apeki o uno de rils mientras se realizaba el de tomate, mostraron que si bien crear dos HDF5 a la vez incrementa el coste de tiempo, este tiempo no es el doble. En el caso de ril tardo 10 segundos en lugar de 8 y en el caso de los dos apeki tardaron 6 minutos y 30 segundos en un caso y 6 minutos y 39 segundos en el segundo frente a los casi 6 minutos que tarda usualmente. De esto se deduce que se puede mejorar el tiempo de dos formas. Una artificialmente si se tienen que pasar dos VCF a HDF5, se pueden transformarlos los dos a la vez en menos tiempo que haciendo uno y luego el otro. La otra, la más adecuada, es buscar una mayor optimización del tiempo, pues esto resulta curioso y podría indicar que Variation 5 puede ser mucho más rápido.

## 4.4. FILTROS_de SNPs

La librería allel no cuenta con ningún método de filtrado lo cual suponía tener que aplicar los filtros antiguos antes de transformar el VCF a HDF5 con lo cual continuábamos con un problema de tiempo. Para solventar esto, se creó un script a fin de filtrar los datos genotype. Este script cuenta también con el cálculo de los puntos del ACP y la creación de las gráficas tridimensionales correspondientes (esta última función realmente es importada de otro script, pero podría haber sido creada también en el mismo), a fin de poder representar los puntos desde distintos ángulos y poder montar una animación rotatoria.

Los filtros se crearon usando NumPy debido a que de este modo teóricamente debe obtenerse una velocidad de filtrado mayor que usando for en Python.

### 4.4.1 Filtros disponibles

Tras diversas pruebas y optimización del código, los filtros de los cuales dispone el script son los siguientes:

-filter_snps_by_maf: Transforma los datos genotype a frecuencias alélicas usando la librería allel. Se crea una matriz de filtrado dándole un valor máximo de frecuencia alélica y esta matriz de booleanos (True si el SNP pasa el filtro y False si no lo hace) se aplica a la de genotypes quedándose solo con aquellos SNP que tengan

frecuencias alélicas inferiores a la deseada, es decir aquellas que sean menores al número que le haya sido asignado a la variable maf (se llama maf como abreviatura de máximum allele frequency).

-filter_snps_by_missing_calls: Cuenta cuántos alelos hay en total por SNP y cuantos pueden haber como máximo (ploidía* número de muestras). Resta al número máximo de alelos, la cantidad que se permita que falten y usa este resultado para compararlo con cuantos tenemos por SNP obteniéndose la matriz de booleanos para filtrar genotype.

-filter_snps_by_min_calls: Cuenta cuántos alelos hay en total por SNP y compara con el número que queramos como mínimo que haya por SNP para obtener la matriz de filtrado.

-filter_gn: Elimina aquellos SNP que una vez una la función to_n_alt() solo contienen el valor 2. Este filtro es necesario para el cálculo del ACP (da error si una fila contiene sólo 0 o 2). No es necesario que este filtro trate de filtrar las filas que solo contengan 0 porque si se ha aplicado el filter_snps_by_maf, no debería aparecer ninguna.

### 4.4.2 NumPy vs for de Python

La primera opción que a uno se le pasa por la cabeza a la hora de seleccionar unas líneas u otras según cumplan una condición o no, es pasar las líneas una a una con un 'for' y luego usar un condicional.

Con esta idea y teniendo en cuenta que se debía mantener la estructura de NumPy array, el primer código creado para filtrar por frecuencia alélica, una vez obtenida con la librería allel es el siguiente:

```
gn_filt = None

for freq in af:


    d = af[freq < 0.95]

    if gn_filt is None:

        gn_filt=np.array(d)

    else:

        gn_sig = np.array(d)

        gn_filt = np.vstack((gn_filt, gn_sig))
```

Este código lo que hace es asignarle el valor *None* a la variable 'gn_filt', ya que en un primer momento no hay nada filtrado. Tras esto se entra en el *for*, que va dando las frecuencias alélicas de cada SNP contenidas en la variable 'af' una a una. En la variable 'd' se guarda cada vez las frecuencias que toque en esa ronda si superan la condición de ser inferior a 0.95.

El condicional if sirve únicamente para la primera vez que se entra en el *for*, para que la variable 'gn_filt' pase de tener nada a tener el primer SNP que ha superado el filtro. En el primer SNP que entre al *if* ya no entrara en el *else* de modo que se volverá a la primera línea del *for* y se repetirá el proceso hasta que se terminen los SNP, solo que ahora los SNP que superen la condición entrarán en el *else* donde se añadirán a 'gn_filt'.

np.array(d) simplemente es para volver a transformar en un array de NumPy y np.vstack para ir añadiendo los SNP que han superado el filtro al array de numpy en construcción que hay dentro de 'gn_filt'.

Tras este se tuvo que crear un segundo filtro que filtrase el array gn para poder calcular el ACP, siguiendo la misma filosofía:

```
for linea in gn:
        if (2 and 0 in linea) or 1 in linea:



            num = len(linea)
            suma = np.sum(linea)
            div = suma/num
            if div<0.005 or div>1.995:
                continue


            if gn_filt is None:
               gn_filt=np.array(linea)
            else:
               gn_sig = np.array(linea)
               gn_filt = np.vstack((gn_filt, gn_sig))
```

En este filtro, se pasa línea a línea cada fila de 'gn'. Si la línea tiene doses y ceros o incluye algún uno continua, si no se pasa a la siguiente fila de 'gn'. Con esto se eliminan todas aquellas filas que contienen únicamente 0 (todos son como el genotipo

de referencia) o únicamente 2 (ninguno es como el genotipo de referencia) que impedirian el cálculo del ACP.

Todas las filas que pasan este primer filtro, deben pasar por un segundo filtro que elimina aquellos que tengan una mayoría de 0 o de 2. Lo que se hace es contar cuantas muestras hay (num), sumar el valor de todos los números en cada fila (suma) y dividir la suma entre num. Si este resultado da menos de 0.005 o más de 1.995 se descarta esa fila y se vuelve al principio del *for* con la siguiente fila de 'gn'. Si pasa este filtro se une a la variable 'gn_filt' del mismo modo que con el filtro de frecuencias alélicas.

Sin embargo, el hecho de usar NumPy directamente, sin necesidad de usar *for* e ir reconstruyendo el array, se planteó como una alternativa interesante en el proceso de optimizar el código ya que en teoría su velocidad debería ser superior ( y así resulta tal y como puede verse en el correspondiente apartado de resultados)

### 4.4.3 Velocidad filtros

Tal y como se puede ver en el anexo tiempos filtros, la diferencia entre filtrar usando el código pensado en primer momento para filtrar por frecuencia alélica (usando for y np.vstack) y el segundo (que usa únicamente NumPy) es despreciable en el caso del fichero más pequeño. Cuando se usa el HDF5 de apeki, la diferencia ya empieza a ser importante, puesto que se pasa de tardar menos de 1 segundo usando tanto un único filtro como los cuatro, a tardar casi tres minutos usando el filtro que usar el for y va montando el array en cada vuelta. Si se compara el tiempo que tarda el filtro realmente con lo que tarda si solo fuese iterando las filas podemos ver que, aunque el for sigue haciendo per se más lento el filtro, lo que realmente ralentiza el programa parece ser montar el array de NumPy paso a paso con el np.vstack. Esta diferencia sola ya es más que suficiente para dejar claro que usar filtros aprovechando en mayor grado las características de NumPy es la mejor opción.

Al comparar la velocidad en el caso del tomates.HDF5, se puede comprobar que usar NumPy para filtrar no solo es una opción claramente más óptima en tiempo, si no que se vuelve una opción necesaria. Mientras que filtrar con los filtros definitivos cuesta menos de 20s, en el caso del otro filtro obtenemos un valor superior de minutos. Siendo los dos valores obtenidos, el primero de 16 minutos y el segundo de 90 minutos, cabe decir que esos tiempos son los que se dejaron correr el script antes de detenerlo. No es necesario saber cuánto tarda realmente el segundo filtro, ya que va a tardar como mínimo 90 minutos y los filtros finales pueden hacer lo mismo o más (usando 4 filtros) en 12 segundos.

Por otro lado está el filtro gn. En este caso el filtro sigue siendo significativamente más lento incluso que usar los cuatro filtros finales. Sin embargo, al compararlo con el otro filtro que también usa for y np.vstack obtenemos que en el caso del fichero menos SNP (ril) tarda un tiempo similar. En el segundo fichero con menos SNP (apeki) tarda un poco más del doble y en el caso del fichero con más SNP (tomatoes) tarda una media de medio minuto, cuando el filtro af tarda como mínimo 90 minutos.

Una posible explicación a esta abismal diferencia de tiempo entre dos filtros que usan una mecánica muy similar es que filtran matrices con distintas dimensiones. La matriz

de frecuencias alélicas en el caso del tomate es de (868978,  22) mientras que la matriz de gn tiene estas dimensiones (868978,  348). Esto  descarta que realmente la diferencia de tiempo sea debida al distinto tamaño de las matrices. La segunda opción es que el filtro gn sea un filtro mucho más potente y deje un número muy inferior de filas que luego tengan que ser unidas de nuevo mediante np.vstack (que tal y como se ha señalado antes, causa un incremento mayor de tiempo que el for en sí mismo). Esta puede ser realmente la causa ya que  los SNPs de gn tras el filtrado son mucho menores: (256, 348). De hecho, si se quita una parte del código del filtro, la cual se podría considerar un segundo filtro dentro del filtro, dejando que sean eliminadas únicamente aquellas filas que contengan únicamente doses o únicamente ceros, no se filtra tanto y tenemos el mismo problema que con el filtro af. Por tanto la lentitud a la hora de filtrar se debe principalmente al número de filas que haya que volver a montar dentro del nuevo array de NumPy filtrado.

## 4.5 Rendimiento del ACP:

### 4.5.1 Velocidad respecto al número de muestras

A fin de ver cuánto tardaba el ACP según se hiciese el cálculo con una cantidad de muestras y si mostraba un comportamiento lineal, se midió el tiempo con una cantidad variable de muestras con una cantidad fija de SNPS.  En (tiempos ACP) se puede ver cuánto se tarda en calcular el ACP, en las diversas condiciones.

El tiempo en las tablas es el tiempo que tarda desde obtener los genotipos del HDF5 hasta calcular los puntos del ACP. Por tanto el tiempo no es únicamente el del ACP, también el de los filtros, lo cual supone que el tiempo real del ACP es de entorno a 12-18s inferior, de acuerdo a los tiempos vistos en el apartado anterior.

Fijándose en los datos de las tablas del anexo tiempos_ACP, se aprecia claramente que no son exactamente lineales, pero se podría asemejar bastante a una recta, y es lo que sucede cuando se hace, tal y  como se puede observar en las  figuras  10 y 11 . Con lo cual se puede afirmar que el tiempo de cálculo del ACP tanto para 100000 y 150000 se puede considerar lineal con el número de muestras hasta al menos 348 muestras.

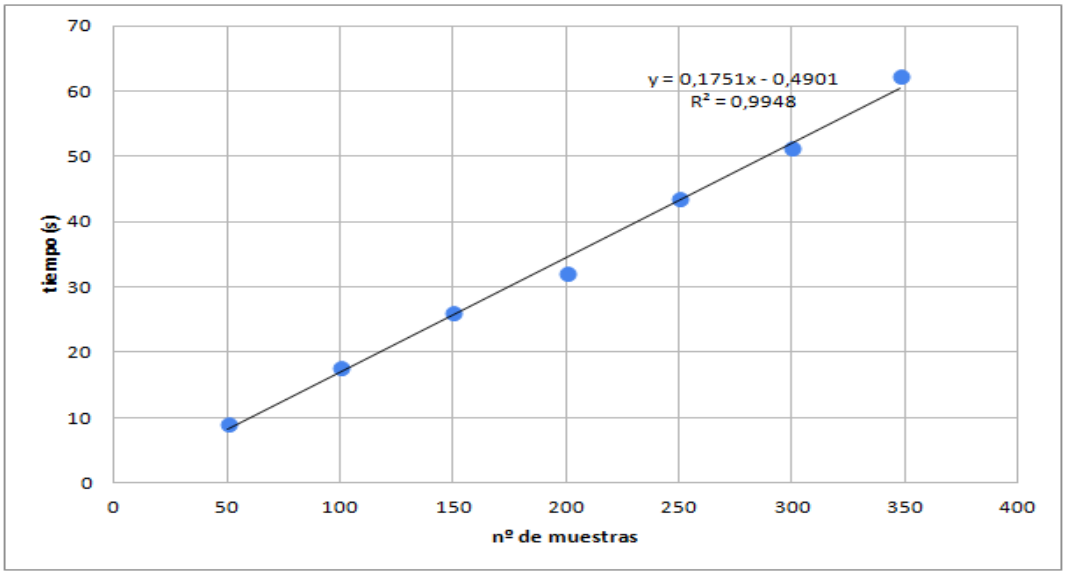


Figura 10: gráfica del tiempo de cálculo del ACP según el número de muestras para 100000 SNPs

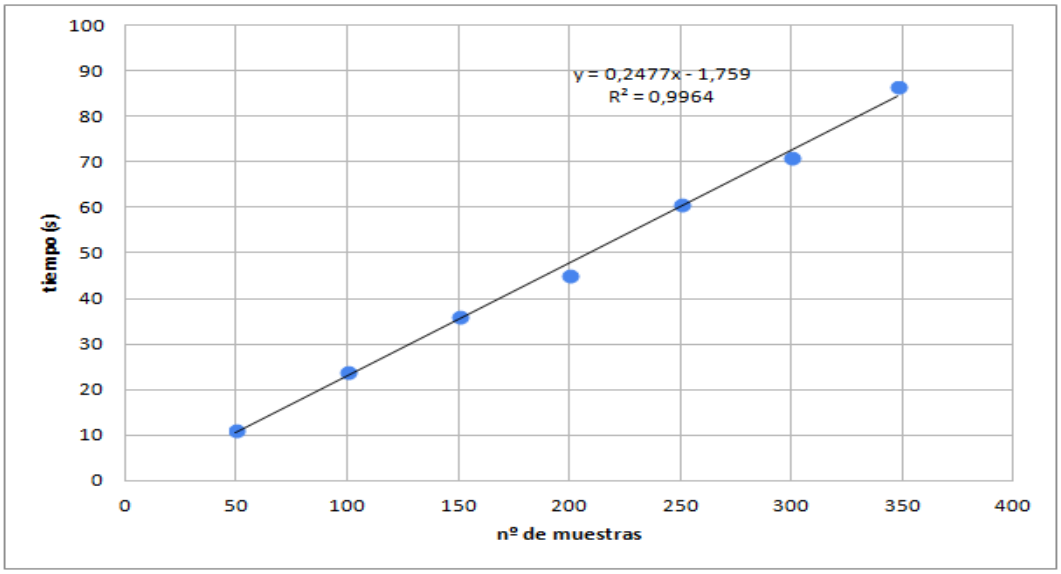


Figura 11: gráfica del tiempo de cálculo del ACP según el número de muestras para 150000 SNPs

### 4.5.2 Velocidad respecto al número de SNP

Por otro lado tenemos para las 348 muestras cuánto tarda en calcular según el número de SNPs.

En la figura 12 se puede ver como con 348 muestras en un ordenador de las características dadas, tenemos una tendencia lineal respecto al incremento de SNP hasta alcanzar los 350000 SNP. A partir de los 400000 SNP ya aparece clara que la tendencia es exponencial (figura 13) hasta que la memoria RAM es incapaz de soportar tantos datos. Este valor máximo de SNP para 348 muestras se encuentra entre los 600000 y los 650000 SNP. Además los tiempos tomados con las cantidades

más altas oscilan más en las distintas medidas, con lo que se hace difícil intentar un tiempo medio que se acerque mucho al real de una ejecución en cualquier momento.

No obstante, esta limitación no supone un problema puesto que tras hacer un filtrado adecuado el número de SNP desciende. En el caso del fichero (tomatos.vcf) pasamos de tener 868978 SNP a tener 868129 tras filtrar por maf=1 y 13687 si permitimos que haya 100 datos faltantes.

Si en lugar de maf=1 se usa maf=0,9 y se permite que como mucho falten 10 datos el número de SNP que  quedan es de tan solo 91014 al pasar el primer filtro, y 28 al pasar el segundo.

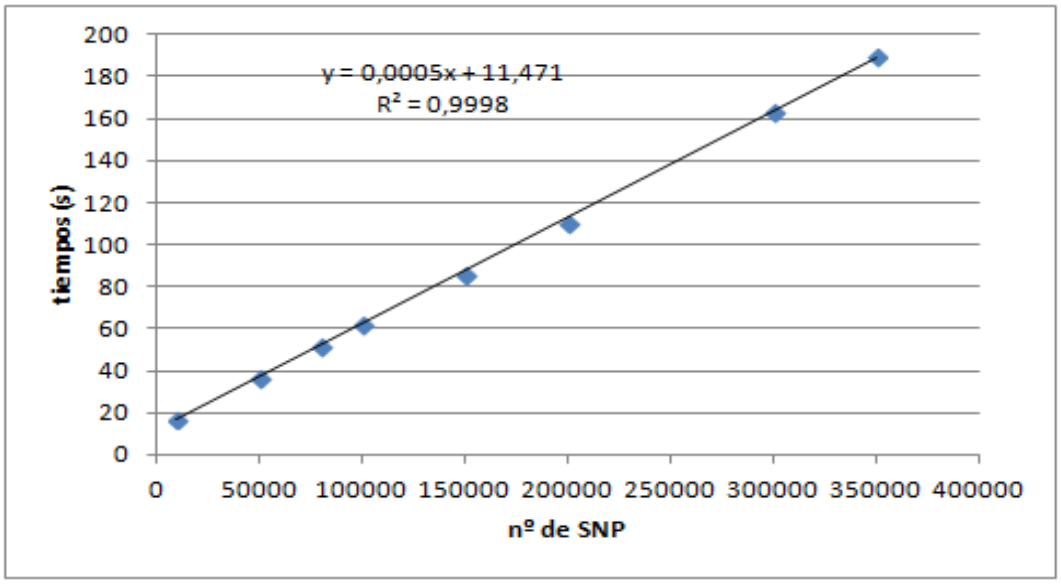


Figura 12: gráfica del tiempo de cálculo del ACP según el número de SNP

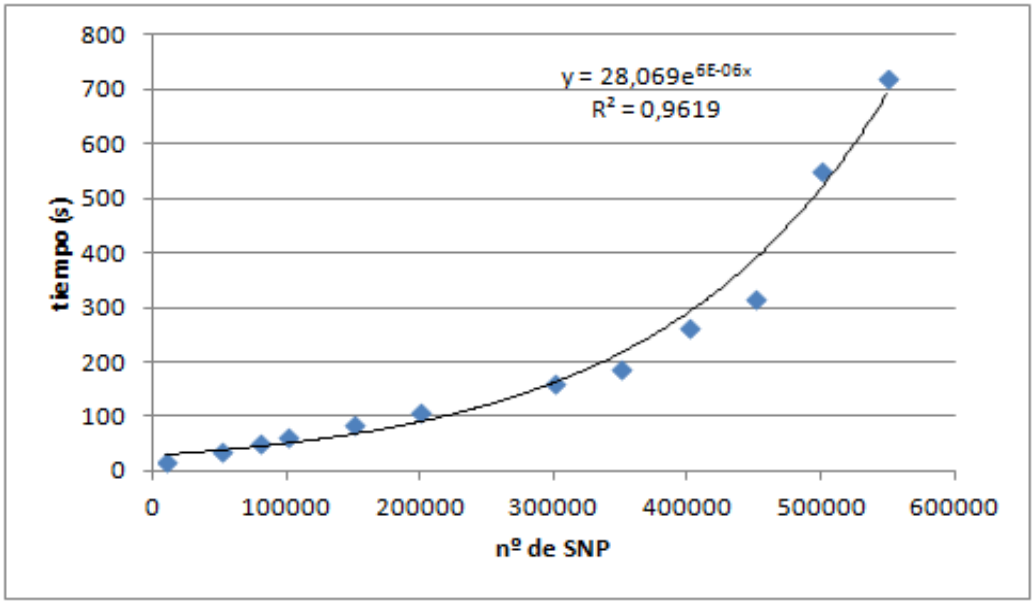


Figura 13: gráfica del tiempo de cálculo del ACP según el número de SNP

Hay que añadir que durante la creación del script filtros_ACP.py y las pruebas realizadas durante esta, se vio que la cantidad máxima de SNP disminuía al ampliar las funciones del script. El motivo es que había más variables que contenían arrays. Se trató de que hubiera el mínimo número de variables con grandes matrices de información. También se borraron las variables que contenían arrays que no iban a ser utilizados de nuevo. De este modo el ordenador dispone de más memoria RAM libre, pudiendo llegar a la cantidad de SNP anteriormente mencionada.

En el anexo velocidades, se puede ver como no borrar esa información hace que cueste más el cálculo. No afecta a los primeros 100000 SNP, pero si en los 200000 y en adelante (lo cual hace que la pendiente de la figura 14 sea mayor a la de la figura 12) hasta llegar al límite de SNP que es inferior ya que se encuentra entre los 450000 y los 500000 SNP.

Este hecho, que podía haber pasado desapercibido de haber hecho las pruebas una vez creadas todas las funciones, es debido a que en Python el manejo de la memoria es automático. Esto normalmente es ventajoso al ahorrar la molestia de hacerlo, en este caso ha hecho que el depender del Garbage collector para eliminar lo que ya no se usa límite nuestro programa por un lado en velocidad y por otro lado en cantidad de SNP máximo. Si bien ya se ha dicho que no se van a alcanzar estos números tras el filtrado, cabe suponer que si se disponen de más muestras el límite se alcanzará antes. No solo esto, sino también que al depender de la RAM, optimizar el uso que hace el script permite disponer de un extra de memoria desocupada que puede usar el ordenador para realizar otras tareas.

En cualquier caso estos números son bastante buenos. Se ha conseguido trabajar con datos de VCF tanto pequeños como grandes en poco tiempo. La única desventaja es tener que invertir tiempo para obtener el HDF5.

Actualmente hay ordenadores pre montados con entre 4 y 12 GB de RAM por menos de 1000€, lo cual significa que cualquier grupo que compre ordenadores va a poder obtener resultados mucho más rápido aún sin necesidad de recurrir a un superordenador

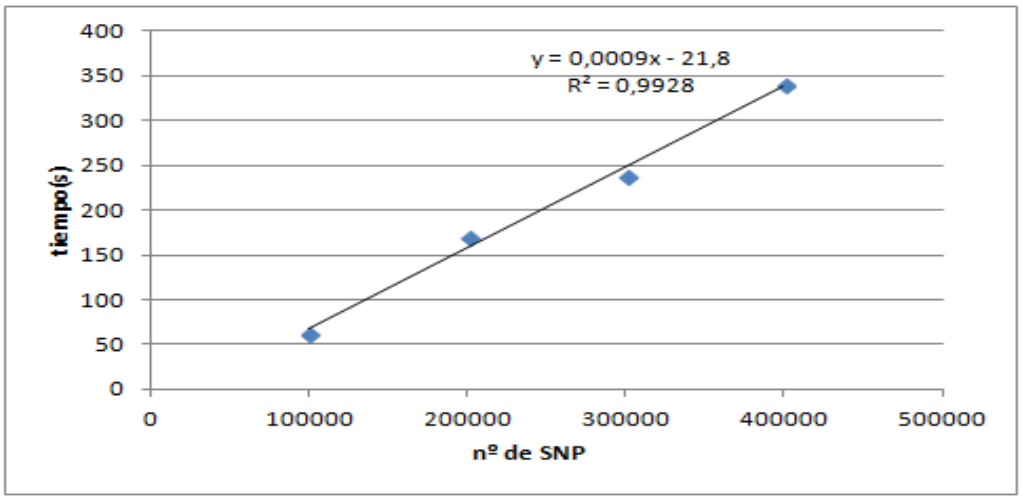


Figura 14: gráfica del tiempo de cálculo de ACP según el número de SNP, sin borrar arrays.

## 4.6 Ejemplos de análisis de datos

### 4.6.1 ACP

En el anexo de los videos se pueden ver las distintas animaciones, según los datos de que parten, si tienen o no ejes y si los tienen si está el nombre o no de los componentes.

#### 4.6.1.1 Tomato

Los primeros videos en ser creados fueron con los datos de tomate. Con los que se probó la velocidad total de filtros_ACP.py en un ejemplo práctico.

Para este ejemplo y los siguientes, se filtró por frecuencia alélica, eliminándose los SNP en los que fuese 1, se filtraron aquellos a los que les faltaban más de 30 datos y como siempre aquellos en el que en el array de gn, solo tuviesen el número 2 para todas sus muestras. 3021 fueron los que pasaron el filtro.

En el caso del tomate se creó un video con imágenes cada 10 ángulos de giro, obteniéndose 36 imágenes. Esto costó tan solo 20s en la primera ejecución (en posteriores pruebas llegó a subir a 30s y a bajar a 15s). Montar la animación con avconv costo tan solo 0,6s.

A parte se crearon imágenes por cada ángulo de giro, obteniéndose 360 imágenes para cada animación. Este es el número de imágenes que van a tener todos los demás videos. En este caso, al tener que crearse un número mayor de imágenes tarda más, alcanzando los 45s y los 6s en la creación de la animación.

Al visualizar el video o ver la figura 15, se ve claramente como el componente 2, separa 3 de las muestras de del resto. El componente 1 separa estas mismas tres muestras junto a otras 5 del resto. En el eje del componente 3 se encuentran más dispersos y no hay una separación clara. Por último se puede ver un cúmulo de puntos muy unidos donde deben concentrarse más de 300 muestras, dicho cúmulo se encuentra desplazado levemente a la derecha de la segunda mitad del eje del componente 2, aproximadamente en el centro del 3 y en la parte del borde derecho del componente uno.

Esto es cuanto se puede extraer de aquí. Se podría tratar de interpretar diciendo que por ejemplo esas más de 300 muestras pertenecen a la misma planta o plantas del mismo invernadero y las otras de dos o tres invernaderos distintos. Sin embargo, esto sería aventurarse mucho puesto que desconozco de donde proceden siquiera las muestras.

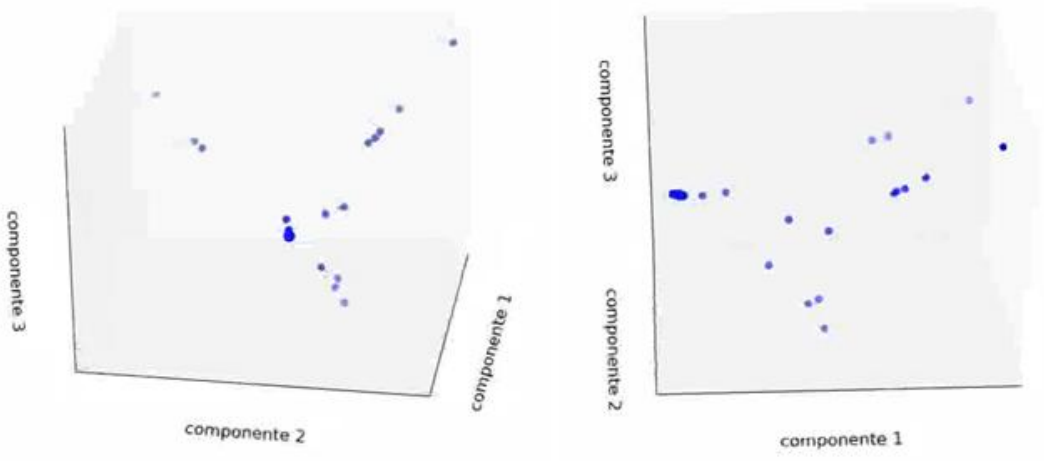


Figura 15: Imágenes de la representación del ACP de tomates desde dos ángulos distintos

#### 4.6.1.2 Apeki

36835 SNP fueron los que pasaron los filtros en el caso de apeki.

En este caso se puede ver en la figura 16 al menos cuatro grupos. Los dos primeros son dos puntos solitarios, separados únicamente por el componente 3, luego otro grupo de 7 u 8 muestras separadas por el componente 1 y por último otros dos puntos separados por el componente 1 entre ellos  y de los demás.

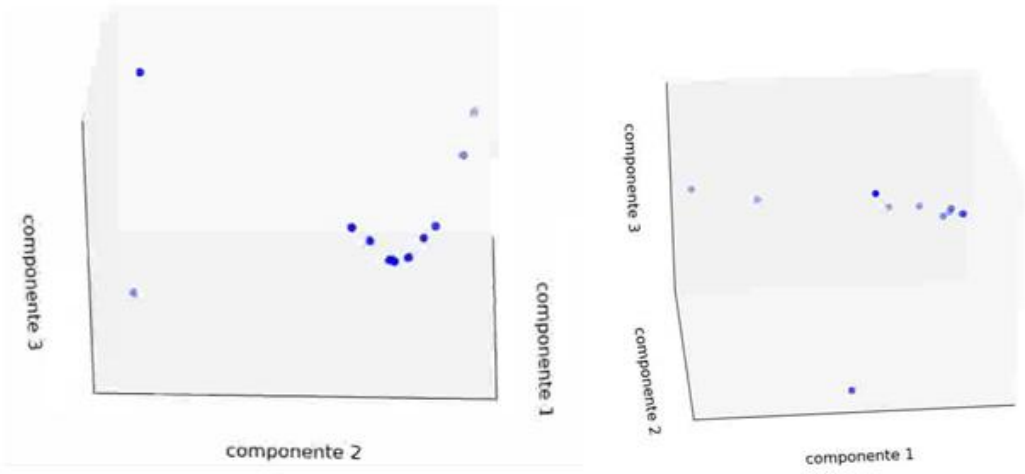


Figura 16: Imágenes de la representación del ACP de apeki desde dos ángulos distintos

#### 4.6.1.3 RIL

En este caso 237 SNP, pasaron los filtros

En este caso como se puede ver en la figura 17 los puntos ocupan una gran zona, creándose nubes de puntos.

El componente 1 separa las muestras en tres grandes bloques, el más grande a la izquierda, el segundo a la derecha y uno más pequeño en la zona central. El componente 3 a su vez separa el grupo más grande subgrupos, siendo difícil decir si en tres o cuatro. El grupo central se encuentra disperso dentro del componente 3 y el grupo de la derecha se divide en un subgrupo grande y en 2-3 más pequeños.

Por último el componente parece separar en dos el grupo más grande, aunque no de forma clara.

Aprovechando que en estás animaciones los puntos están muy dispersos, se creó un video asignando distintos colores y formas a grupos de muestras a fin de mostrar que si se quiere diferenciar a simple vista muestras de un tipo de otras es posible. En este caso como no hay motivo para separar las muestras de este modo más allá de servir de ejemplo, los colores y formas se asignaron cada 20 muestras en el orden en el que están. Este video es el llamado ril_colorin.avi.

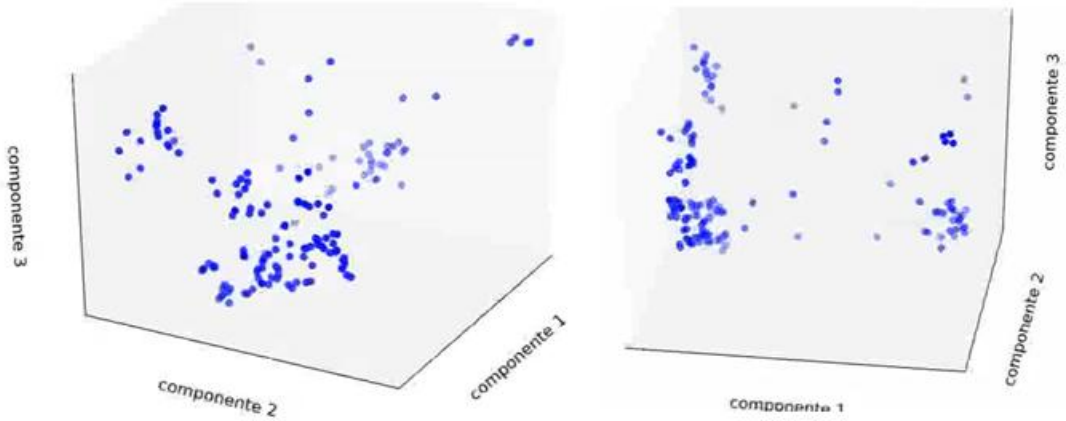


Figura 17: Imágenes de la representación del ACP de ril desde dos ángulos distintos

## 4.7 Discusión general

La librería VCFNP ha mostrado ser útil para un determinado rango de tamaño de ficheros VCF para un ordenador con una RAM determinada. Para ficheros con una cantidad mayor de datos de genotipado de ese rango, no funciona. Para solucionarlo la creación de la librería Variation 5 ha mostrado ser útil, ya que el tamaño del VCF no resulta una limitación a la hora de poder usar o no esta librería. Además ahora es posible guardar información que no era guardada con la librería VCFNP. Además guarda la información en un único archivo HDF5. La librería Variation 5 ha mostrado también tener una mayor compresión y una mayor velocidad a la hora de crear HDF5 a partir del VCF más grande de este trabajo. Por el contrario con los VCF más pequeños usados es más lento usar Variation 5, lo cual indica que todavía puede ser mejorable en ese aspecto. Otro punto a mejorar de esta librería es incluir más funciones para el manejo de dato de genotipado o crear estás funciones en una librería nueva.

Para esas funciones que trabajen con los datos de genotipado, la librería allel a mostrado ser muy útil, pero es posible que al realizar un análisis más exaustivo de los datos se echen en falta algunas funciones que habría que crear en ese momento.

Los filtros creados usando NumPy han mostrado ser lo suficientemente rápidos para poder filtrar cerca de un millón de SNPs en cuestión de segundos. Sería interesante crear filtros adicionales para que el usuario disponga de mayores opciones de filtrado.

En definitiva, se han logrado los objetivos, pero se podría optimizar la librería Variation5 y crear herramientas adicionales.

# 5. CONCLUSIONES

-Hemos comprobado que HDF5 es un formato de archivo que permite trabajar con grandes cantidades de datos matriciales utilizando una cantidad de memoria limitada.

-A pesar de utilizar el disco, HDF5 permite hacer análisis eficientemente, puesto que acceder a la información de un HDF5 es más rápido que acceder a la misma información en un fichero VCF.

-Para realizar análisis sobre las matrices de datos es preferible utilizar los métodos de cálculo disponibles para los arrays de NumPy frente a los implementados línea por línea en Python. Estos últimos son mucho más lentos debido a la lentitud de Python frente a la implementación en C de las funciones de NumPy

-Crear ficheros NumPy sólo es posible si podemos almacenar todos los datos a la vez en memoria. Esta limitación es superada por la librería Variation5 que permite tratar las líneas de los ficheros VCF en bloques

-Cualquier ordenador con 3GB de RAM es capaz de realizar en cuestión de minutos el ACP y representarlo desde distintos ángulos, de más de 100000 SNP y más de 300 muestras, si tiene dicha información en un HDF5.

-El tamaño de los archivos HDF5 obtenidos con Variation 5 frente al tamaño de los VCF, muestran que es posible ahorrar disco.

-Dada las ventajas de los ficheros HDF5 podría recomendarse como buen sistema para guardar datos de genotipado.

# 6. BIBLIOGRAFÍA


ABDI, H. WILLIAMS, J. L. (2010)*Principal component analysis.* Wires Comp Stat, 2 : 433–459

ALIMANFOO (ALISTAIR MILES), 2015. *vcfnp*, visto el 29 de abril de 2015 **https://github.com/alimanfoo/vcfnp**

ALISTAIR MILES, 2015. *scikit-allel - Explore and analyse genetic variation*, visto el 11 de mayo de 2015

**http://scikit-allel.readthedocs.org/en/latest/index.html**

ANDREW COLLETE, 2014. *HDF5 for Python*, visto el 30 de abril de 2015. **http://docs.h5py.org/en/latest/index.html**

COLLETE, A. (2014). *Python and HDF5.* Ed. O'Reilly Media. Sebastopol.

CORNELL, 2013. *Genotyping by sequencing (GBS) Method overview,* visto el 17 de Julio de 2015
http://cbsu.tc.cornell.edu/lab/doc/GBS_Method_Overview1.pdf

DAVEY, W. J. HOHENLOHE, A. P. ETTER, D. P. BOONE, Q. J. CATCHEN, M. J. AND BLAXTER, L. M. (2011). Genome-wide genetic marker discovery and genotyping using next-generation sequencing. Nature Reviews Genetics, 12: 499-510

Donato, D. M. Peters, O. S. Mitchell, E. S. Hussain, T. Imumorin, G. I. (2013). Genotyping-by-Sequencing (GBS): A Novel, Efficient and Cost-Effective Genotyping Method for Cattle Using Next-Generation Sequencing. Plos one.

http://cbsu.tc.cornell.edu/lab/doc/GBS_Method_Overview1.pdf

ILLUMINA, 2015. *An introduction to Next-Generation Sequencing Technology,* visto el 16 de Julio de 2015
http://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf

ILLUMINA, 2015. *Infinium HD Assay,* visto el 17 de julio de 2015
http://www.illumina.com/technology/beadarray-technology/infinium-hd-assay.html

LIBAV, 2015. *avconv documentation*, visto el 18 de junio de 2015 **https://libav.org/documentation/avconv.html**

LIFE TECHNOLOGIES, 2015. *Personal Genome Machine® (PGM™) System,* visto el 16 de julio de 2015.
http://www.lifetechnologies.com/order/catalog/product/4462921

MATPLOTLIB, 2015.*matplotlib*, visto el 25 de mayo de 2015
**http://matplotlib.org/index.html**

MOROZOVA, O. MARRA, A. M. (2008*) Applications of next-generation sequencing technologies in functional genomics.* Genomics, 92: 255-264

PACIFIC BIOSCIENCES, 2014. *SMRT technology,* visto el 16 de julio de 2015.
http://www.pacificbiosciences.com/products/smrt-technology

PATRICK, K. (2007). 454 Life Sciences: Illuminating the Future of Genome Sequencing and Personalized Medicine. Yale Journal of Biology and Medicine,80: 191-4

SANGER, F. AND COULSON, A. R. (1975). A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. Journal of Molecular Biology, 94: 441-448.

SCIKIT-LEARN, 2014. *sklearn.cluster.KMean,* visto el 19 de junio de 2015
**http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans**

SCIPY, 2013. *Cookbook*, visto el 13 de mayo de 2015.
**http://wiki.scipy.org/Cookbook**

WIKIPEDIA, 2015. *Hierarchical Data Format*, visto el 29 de abril de 2015.
**https://en.wikipedia.org/wiki/Hierarchical_Data_Format**

WIKIPEDIA, 2015. *Análisis de componentes principales*, visto el 20 de mayo de 2015
**https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales**

WIKIPEDIA, 2015. *Polimorfismo de nucleótido simple*, visto el 10 de julio de 2015
**https://es.wikipedia.org/wiki/Polimorfismo_de_nucle%C3%B3tido_simple**

1000 Genomes, 2013. *The Variant Call Format (VCF) Version 4.1 Specication*, visto el 1 de mayo de 2015
http://samtools.github.io/hts-specs/VCFv4.1.pdf

# 7. ANEXOS

**GLOSARIO**

ACP: Es una técnica usada para reducir las dimensiones de un conjunto de datos, utilizada para facilitar la búsqueda de la causa de la variabilidad de los datos.

Avconv: Librería que permite la edición de video y audio.

Datasets: Estructura del HDF5 que contiene información, usualmente matrices de números.

Groups: Estructura del HDF5 que puede contener otros *groups* o *datasets.*

HDF5: 1. Hierarchical Data Format. Formato de archivo caracterizado por estructurar la información usando *groups* y *datasets*. 2. Librería que permite leer y crear archivos con formato HDF5.

Indels: Inserciones y deleciones.

Librería: Conjunto de scripts que componen un determinado programa.

Matplotlib: Libreria de Python para la creacion de distintos tipos de graficas en 2D y 3D.

NumPy: Numeric Python.Librería para Python escrita en C, C++ y Fortran que contiene diversas funciones para la creacion de matrices y cálculos matemáticos.

Python: Lenguaje de programación diseñado en 1991 por Guido Van Rossum.

Script: Archivo de texto plano que contiene las órdenes (código) para realizar alguna tarea. Pueden ser considerados mini programas.

SNP: Single Nucleotide Polymorphism. Cambio de un nucleótido en un punto del genoma. A veces los indels se engloba en este término.

Ubuntu: Sistema operativo libre basado en GNU/Linux.

Variables: En programación, un conjunto de caracteres (usualmente una palabra que recuerde su contenido) que contiene la información que se le haya asignado.

VCF: Variant Call Format. Formato de archivo que almacena información sobre SNP e indels de varias muestras de una determinada especie o variedad con respecto al genotipo de referencia.

**FILTROS_ACP.PY**

```
import h5py
import allel
import numpy as np

#For create the graphic
from draw_ACP import draw_ACP
import sklearn
from sklearn.cluster import KMeans




def open_HDF5(filename):
	h5_file = h5py.File(filename, 'r')
	return h5_file


def extract_genotype_data(h5_file):

	'''Transform the genotype data in h5_file to a GenotypeArray'''


	genotypes = h5_file['calldata']['genotype']
	#genotypes = h5_file['calldata']['GT']
	return allel.model.GenotypeArray(genotypes)




def filter_snps_by_maf(genotype_array, max_freq=1):

	'''Eliminates the SNP with a allelic frequencie bigger than max_freq'''

	af = genotype_array.count_alleles().to_frequencies()
	maf = np.amax(af, axis=1)
	is_polimorf = maf < max_freq
	return genotype_array[is_polimorf, :, :]


def calculate_total_alleles(genotype_array):

	'''With the genotype array, guess the ploidy and the number of samples to calculate'''
	'''how many alleles per SNP have the array'''

	g_ploidy = genotype_array.ploidy
```

```
    g_samples = genotype_array.n_samples
    return g_ploidy*g_samples


def filter_snps_by_missing_calls(genotype_array, num_al_snp, max_missing):

    '''Eliminate SNP whith less missing data than max_missing'''
    '''num_al_snp is the total number of alleles per SNP'''

    al = genotype_array.count_alleles()
    suma = np.sum(al, axis=1)
    min_all = num_al_snp-max_missing
    filt = suma > min_all

    return genotype_array[filt, :, :]


def filter_snps_by_min_calls(genotype_array, min_calls):

    '''Eliminates SNP whith less data than min_calls'''

    al = genotype_array.count_alleles()
    suma = np.sum(al, axis=1)
    filt = suma >= min_calls
    return genotype_array[filt, :, :]



def filter_gn(gn, num_al_snp):

    '''Eliminates the rows with only 2'''
    '''This filter is requiered if you have any row whith all 2, because ACP falls'''
    '''Works with ploidy==2'''
    suma = np.sum(gn, axis=1)
    div =suma/num_al_snp
    filt = div<1 #If all are 2, div will be 1.
    return gn[filt, :]



def filter_(genotype_array, filters=True, num_al_snp=None, max_missing=0,
min_calls=1):

    '''Uses at least maf filter'''
    '''If filters==2 filts with missing call, 3 with min calls, and 4 with both'''

    if filters:
    g_filt = filter_snps_by_maf(genotype_array)

    if filters==2:
```

```
        g_filt = filter_snps_by_missing_calls(g_filt, num_al_snp, max_missing)
        elif filters==3:
        g_filt = filter_snps_by_min_calls(g_filt, min_calls)
        elif filters==4:
        g_filt = filter_snps_by_missing_calls(g_filt, num_al_snp, max_missing)
        g_filt = filter_snps_by_min_calls(g_filt, min_calls)

        return g_filt




def obtain_ACP_points(ACP):
        '''Obtain axes X, Y, Z like the first, second and third components'''

        X = []
        Y = []
        Z = []
        valores=ACP[0] #ACP[0] are point data, ACP[1] is the object
        for i in range(len(valores)):
        X.append(ACP[0][i][0])
        Y.append(ACP[0][i][1])
        Z.append(ACP[0][i][2])
        #i is the sample number
        #The third number is the component


        return X, Y, Z

def calculate_Kmeans(gn, n_clusters):

        KM_cluster = KMeans(n_clusters=n_clusters)
        KM_fit = KM_cluster.fit(gn)
        centr = KM_fit.cluster_centers_
        X, Y, Z = centr[0], centr[1], centr[2]

        return X, Y, Z

def main(filename=None, filters=None, max_freq=1):

        filename = '/home/felipe/Documentos/HDF5/ril_call2d.HDF5'
        #filename = '/home/felipe/Documentos/HDF5/calldata_2d_apeki.HDF5'
        #filename = '/home/felipe/Documentos/HDF5/tom_call2d.HDF5'

        h5_file = open_HDF5(filename)
        genotype_array = extract_genotype_data(h5_file)
        num_al_snp = calculate_total_alleles(genotype_array)
        g_filt = filter_(genotype_array, filters=2, num_al_snp=num_al_snp,
max_missing=30)
```

```
        #g_filt = g_filt[:450000, :, :]
        gn = g_filt.to_n_alt()

        gn = filter_gn(gn, num_al_snp)


        #Deleting the variables with genotypes that we don't need, increases speed and
        #the max number of snp that can calculate the ACP function

        del(genotype_array)
        del(g_filt)


        ACP = allel.stats.decomposition.ACP(gn)


        X, Y, Z = obtain_ACP_points(ACP)

        #X, Y, Z = calculate_Kmeans(gn, n_clusters=5)

        return X, Y, Z

if __name__=='__main__':

        proj = main(filename = '/home/felipe/Documentos/HDF5/tom_call2d.HDF5')
        draw_ACP(proj)
```

**DRAW_ACP.PY**

```
from matplotlib.figure import Figure
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from mpl_toolkits.mplot3d import Axes3D
import os



def draw_ACP(projections, color='blue', dir_ ='.', opt_ax = 'off'):
	X, Y, Z = projections

	fig = Figure()

	axes = fig.add_subplot(111, projection='3d', xticks=[], yticks=[], zticks=[])
	axes.scatter3D(X, Y, Z, color=color)
	axes.axis(opt_ax)

	for i, angle in enumerate(range(0, 360, 1)):
		fname = 'ril_comp_%04d.png' % (i)
		fpath = os.path.join(dir_, fname)
		fhand = open(fpath, 'w')
		if not opt_ax=='off':
			axes.set_xlabel('componente 1')
			axes.set_ylabel('componente 2')
			axes.set_zlabel('componente 3')
		axes.view_init(azim=angle)
		canvas = FigureCanvas(fig)
		canvas.print_figure(fhand)
		fhand.close()
```

TIEMPOS FILTROS

| | | | | Numpy | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | t(s) | tiempo antes primer filtro | filtro maf=1 | filtro maaf=1+ miss=100 | filtro maaf=1+ miss=100+ min calls=1 | filtro maaf=1+ miss=100+ min calls=1+gn | |
| | t1 | 0,94 | 0,745 | 0,745 | 0,691 | 0,794 | |
| | t2 | 0,726 | 0,701 | 0,693 | 0,696 | 0,712 | |
| ril | t3 | 0,743 | 0,684 | 0,703 | 0,703 | 0,708 | |
| | t4 | 0,724 | 0,687 | 0,687 | 0,718 | 0,701 | |
| | t5 | 0,733 | 0,679 | 0,7 | 0,703 | 0,704 | |
| | tmedio | 0,7732 | 0,6992 | 0,7056 | 0,7022 | 0,7238 | |
| | | | | | | | |
| | t1 | 0,759 | 0,738 | 0,773 | 0,737 | 0,751 | |
| | t2 | 0,73 | 0,733 | 0,734 | 0,732 | 0,748 | |
| apeki | t3 | 0,723 | 0,701 | 0,738 | 0,745 | 0,757 | |
| | t4 | 0,726 | 0,769 | 0,743 | 0,737 | 0,752 | |
| | t5 | 0,751 | 0,711 | 0,732 | 0,789 | 0,763 | |
| | tmedio | 0,7378 | 0,7304 | 0,744 | 0,748 | 0,7542 | |
| | | | | | | | |
| | t1 | 4,45 | 27,27 | 13,066 | 12,595 | 11,926 | |
| | t2 | 6,224 | 24,103 | 14,646 | 12,134 | 12,23 | |
| tomate | t3 | 4,49 | 17,549 | 13,726 | 12,966 | 11,972 | |
| | t4 | 4,491 | 9,205 | 12,197 | 12,181 | 12,894 | |
| | t5 | 4,416 | 9,032 | 12,149 | 11,882 | 12,167 | |
| | tmedio | 4,8142 | 17,4318 | 13,1568 | 12,3516 | 12,2378 | |

| | | For y np.vstack | |
|---|---|---|---|
| | | | |
| t(s) | af | af(sin np.vstack) | gn |
| t1 | 1,298 | 0,91 | 0,943 |
| t2 | 0,925 | 0,894 | 0,912 |
| t3 | 0,891 | 0,903 | 0,905 |
| t4 | 0,903 | 0,91 | 0,91 |
| t5 | 0,888 | 0,915 | 0,901 |
| tmedio | 0,981 | 0,9064 | 0,9142 |
| | | | |
| t1 | 170,599 | 1,337 | 2,837 |
| t2 | 166,062 | 1,339 | 2,955 |
| t3 | 167,672 | 1,451 | 2,822 |
| t4 | 167,939 | 1,329 | 2,959 |
| t5 | 168,082 | 1,369 | 2,853 |
| tmedio | 168,0708 | 1,365 | 2,8852 |
| | | | |
| t1 | 16min+(parado a mano (ctrlC) a 16minutos) | 30,973 | 31,51 |
| t2 | 90min+(parado a mano (ctrlC) a 90 minutos) | 28,51 | 31,949 |
| t3 | | 25,253 | 30,676 |
| t4 | | 19,113 | 33,041 |
| t5 | | 19,404 | 34,977 |
| tmedio | | 24,6506 | 32,4306 |

TIEMPOS POR MUESTRAS

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 100k SNP | | | | | | | |
| | | | | | | | |
| nºmuestras | 50 | 100 | 150 | 200 | 250 | 300 | 348 |
| t1(s) | 9 | 17 | 26 | 32 | 43 | 51 | 62 |
| t2(s) | 9 | 17 | 26 | 32 | 43 | 51 | 62 |
| t3(s) | 9 | 18 | 26 | 32 | 44 | 51 | 63 |
| t4(s) | 9 | 17 | 26 | 32 | 43 | 51 | 62 |
| t5(s) | 9 | 19 | 26 | 32 | 44 | 52 | 62 |
| tmedio | 9 | 17,6 | 26 | 32 | 43,4 | 51,2 | 62,2 |
| | | | | | | | |
| 150k SNP | | | | | | | |
| | | | | | | | |
| nºmuestras | 50 | 100 | 150 | 200 | 250 | 300 | 348 |
| t1(s) | 11 | 24 | 36 | 45 | 61 | 71 | 87 |
| t2(s) | 11 | 24 | 36 | 45 | 61 | 71 | 87 |
| t3(s) | 11 | 24 | 36 | 45 | 60 | 71 | 86 |
| t4(s) | 11 | 23 | 36 | 45 | 61 | 71 | 86 |
| t5(s) | 11 | 24 | 36 | 45 | 61 | 71 | 86 |
| tmedio | 11 | 23,8 | 36 | 45 | 60,8 | 71 | 86,4 |

TIEMPOS NPY HDF5

| | | | |
|---|---|---|---|
| | | ril.vcf | |
| tamaño comprimido | 504,5kB | nºmuestras | 153 |
| tamaño descomprimido | 2,5MB | nºSNP | 943 |
| | | vcf2npy (s) | vcfnp2hdf5 (s) |
| | t1 | 1 | 1,335 |
| | t2 | 0,144 | 0,774 |
| variants | t3 | 0,148 | 0,704 |
| | t4 | 0,154 | 0,701 |
| | t5 | 0,163 | 0,773 |
| | tamaño | 223 K | 477K |
| | tamaño comprimido(tar.gz) | 74K | 151K |
| | tamaño comprimido (gz) | 75K | 152K |
| | t1 | 3,581 | 0,707 |
| | t2 | 0,153 | 0,721 |
| calldata | t3 | 0,149 | 0,729 |
| | t4 | 0,148 | 0,744 |
| | t5 | 0,143 | 0,73 |
| | tamaño | 6,4M | 1,5M |
| | tamaño comprimido(tar.gz) | 438K | 1,5M |
| | tamaño comprimido (gz) | 437K | 1,5M |
| | t1 | 3,445 | 0,707 |
| | t2 | 0,183 | 0,701 |
| calldata_2d | t3 | 0,153 | 0,719 |
| | t4 | 0,15 | 0,759 |
| | t5 | 0,152 | 0,756 |
| | tamaño | 6,4M | 1,5M |
| | tamaño comprimido(tar.gz) | 437K | 1,3M |
| | tamaño comprimido (gz) | 437K | 1,3M |
| | | | |
| | suma del tamaño de los 3 ficheros | 13M | 3,5M |

| | | | |
|---|---|---|---|
| | tamaño de los 3 ficheros comprimidos(tar.gz) | 948K | 2,9M |
| | suma del tamaño de los tres ficheros compromidos (gz) | 948K | 2,9M |

| | | | |
|---|---|---|---|
| | | tomate_apeki | |
| tamaño comprimido | 9,7MB | nºmuestras | 12 |
| tamaño descomprimido | 35,6MB | nºSNP | 41682 |
| | | vcf2npy (s) | vcfnp2hdf5 (s) |
| | t1 | 6,528 | 1,211 |
| | t2 | 0,15 | 1,221 |
| variants | t3 | 0,151 | 1,232 |
| | t4 | 0,149 | 1,243 |
| | t5 | 0,146 | 1,271 |
| | tamaño | 9,6M | 4,9M |
| | tamaño comprimido(tar.gz) | 2,7M | 4,5M |
| | tamaño comprimido (gz) | 2,7M | 4,5M |
| | t1 | 14,3 | 2,067 |
| | t2 | 0,152 | 1,841 |
| calldata | t3 | 0,148 | 1,915 |
| | t4 | 0,155 | 1,875 |
| | t5 | 0,15 | 1,912 |
| | tamaño | 22M | 12M |
| | tamaño comprimido(tar.gz) | 4,7M | 12M |
| | tamaño comprimido (gz) | 4,7M | 12M |
| | t1 | 13,9 | 2,165 |
| | t2 | 0,165 | 2,079 |
| calldata_2d | t3 | 0,151 | 2,037 |
| | t4 | 0,151 | 2,17 |
| | t5 | 0,154 | 2,043 |
| | tamaño | 22M | 11M |
| | tamaño comprimido(tar.gz) | 4,7M | 11M |

| | | | |
|---|---|---|---|
| | tamaño comprimido (gz) | 4,7M | 11M |
| | | | |
| | suma del tamaño de los 3 ficheros | 53,6M | 27,9M |
| | tamaño de los 3 ficheros comprimidos(tar.gz) | 12M | 27M |
| | suma del tamaño de los tres ficheros compromidos (gz) | 12M | 27M |

| | | | |
|---|---|---|---|
| | | tomatos | |
| tamaño comprimido | 1,7GB | nºmuestras | 348 |
| tamaño descomprimido | 7,1GB | nºSNP | 868978 |
| | | vcf2npy (s) | vcfnp2hdf5 (s) |
| | t1 | 4m51 | 19,292 |
| | t2 | 3,115s | 19,302 |
| variants | t3 | 0,183s | 19.281 |
| | t4 | 0,144s | 19,391 |
| | t5 | 0,149s | 19,386 |
| | tamaño | 200M | 124M |
| | tamaño comprimido(tar.gz) | 80M | 123M |
| | tamaño comprimido (gz) | 81M | 123M |
| | t1 | 55m23s | 45,231 |
| | t2 | 1,426 | 48,694 |
| calldata (solo GT) | t3 | 0,152 | 42,106 |
| | t4 | 0,142 | 38,956 |
| | t5 | 0,149 | 39,118 |
| | tamaño | 577M | 46M |
| | tamaño comprimido(tar.gz) | 23M | 43M |
| | tamaño comprimido (gz) | 23M | 43M |

| | | | |
|---|---|---|---|
| | t1 | 53m56s | 28,458 |
| | t2 | 1,476 | 35,977 |
| calldata_2d(solo GT) | t3 | 0,151 | 22,815 |
| | t4 | 0,16 | 19,376 |
| | t5 | 0,153 | 18,578 |
| | tamaño | 577M | 46M |
| | tamaño comprimido(tar.gz) | 23M | 44M |
| | tamaño comprimido (gz) | 23M | 44M |
| | | | |
| | suma del tamaño de los 3 ficheros | 1354M | 216M |
| | tamaño de los 3 ficheros comprimidos(tar.gz) | 125M | 209M |
| | suma del tamaño de los tres ficheros compromidos (gz) | 126M | 209M |

VELOCIDADES

| SNP | 10000 | 50000 | 80000 | 100000 | 150000 | 200000 | 300000 | 350000 | 400000 | 450000 | 500000 | 550000 | 600000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| t1(s) | 20 | 36 | 52 | 62 | 87 | 111 | 160 | 188 | 274 | 327 | 541 | 662 | 699 |
| t2(s) | 17 | 37 | 52 | 62 | 87 | 111 | 160 | 192 | 260 | 303 | 550 | 752 | 689 |
| t3(s) | 17 | 37 | 53 | 62 | 86 | 111 | 178 | 189 | 256 | 313 | 526 | 726 | 765 |
| t4(s) | 17 | 36 | 52 | 63 | 86 | 112 | 166 | 191 | 262 | 334 | 551 | 841 | 909 |
| t5(s) | 17 | 38 | 53 | 62 | 86 | 111 | 157 | 189 | 277 | 313 | 590 | 620 | 837 |
| | | | | | | | | | | | | | |
| tmedio (s) | 17,6 | 36,8 | 52,4 | 62,2 | 86,4 | 111,2 | 164,2 | 189,8 | 265,8 | 318 | 551,6 | 720,2 | 779,8 |

| Sin usar del | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| SNP | 100000 | 200000 | 300000 | 400000 |
| t1(s) | 68 | 114 | 226 | 360 |
| t2(s) | 61 | 288 | 250 | 421 |
| t3(s) | 62 | 190 | 246 | 288 |
| t4(s) | 61 | 142 | 237 | 316 |
| t5(s) | 61 | 121 | 233 | 314 |
| | | | | |
| tmedio(s) | 62,6 | 171 | 238,4 | 339,8 |

TIEMPOS INOUT

| Ril | | | | |
|---|---|---|---|---|
| | CHUNK=200 | CHUNK=1000 | CHUNK=200 | CHUNK=1000 |
| shuffle y fletcher | TRUE | TRUE | FALSE | FALSE |
| t1(s) | 8,6 | 8,5 | 8,6 | 8,6 |
| t2(s) | 8,6 | 8,6 | 8,5 | 8,6 |
| t3(s) | 8,5 | 8,5 | 8,6 | 8,5 |
| t4(s) | 8,5 | 8,6 | 8,5 | 8,6 |
| t5(s) | 8,6 | 8,8 | 8,6 | 8,5 |
| | | | | |
| tamaño hdf5 | 517K | 511K | 483K | 485K |

| Apeki | | | | | |
|---|---|---|---|---|---|
| | CHUNK=200 | CHUNK=10000 | CHUNK=25000 | CHUNK=200 | CHUNK=25000 |
| shuffle y fletcher | TRUE | TRUE | TRUE | FALSE | FALSE |
| t1(s) | 357 | 353 | 369 | 353 | 374 |
| t2(s) | 361 | 359 | 364 | 350 | 357 |
| t3(s) | 346 | 355 | 367 | 375 | 363 |
| t4(s) | 348 | 362 | 368 | 390* | 359 |
| t5(s) | 347 | 357 | 373 | 399* | 363 |
| | | | | | |
| tamaño hdf5 | 4.3M | 3.4M | 3.3M | 4.2M | 3.4M |
| | | | | | |
| | *Ejecutadas a la vez | | | | |

| tomatos | | | | |
|---|---|---|---|---|
| kept_field=['GT'] | | | | kept_field=None |

| | CHUNK=200 | CHUNK=200 | | CHUNK=200 |
|---|---|---|---|---|
| shuffle y fletcher | TRUE | FALSE | | TRUE |
| t1(s) | 1813 | 1887 | | 10613 |
| t2(s) | 1907 | 1884 | | 11450* |
| t3(s) | 1873 | 1891 | | 11053* |
| t4(s) | 1886 | 1872 | | 11393* |
| t5(s) | 1914 | 2068* | | 11229* |
| | | | | |
| tamaño hdf5 | 38M | 38M | | |
| | | | | |
| | | | | |
| | *Con Google Drive abierto | | | |

VIDEOS

https://www.dropbox.com/s/fjw0pjup9edpbue/Videos.zip?dl=0