VNIVERSITAT DE VALÈNCIA

Facultat de Ciències Biològiques

Populy

Un paquete en Python para enseñar biología evolutiva

Grado en Biotecnología

Curso 2021/2022

Modalidad: Trabajo de metodologías docentes

Autor:

Mario Ruiz

Tutor:

José Ignacio Lucas Lledó

2022-06-09

Índice

Introducción **3**

Objetivos 6

Computación como herramienta educativa **6**

Alfabetización informática 6

Lenguajes de programación 8

Python 10

Jupyter notebook 10

Métodos de uso libreta Jupyter en el aula 11

¿Cómo utilizar una libreta Jupyter? 12

Genética de poblaciones **13**

Estructura poblacional 14

Evolución y fuerzas evolutivas 15

Mutación 16

Selección natural 16

Migración 17

Deriva genética 17

Equilibrio de Hardy-Weimberg 18

Arquitectura genética 18

Populy: uso y estructura **19**

Métodos de uso 19

Caso práctico: la deriva genética 24

Estructura 26

Perspectiva y limitaciones 28

Conclusiones **28**

Referencias **29**

Appéndice **31**

Introducción

Tanto la genética como la evolución son dos asignaturas fundamentales para cualquier estudiante de ciencias de la vida, estos dos campos de estudio están estrechamente relacionados, normalmente bajo el término de genética de poblaciones. Esta es una rama de la biología que aspira al estudio de la variación de las características genéticas de una población a lo largo de las generaciones (Okasha 2016). Los cambios en los loci, o lugares del genoma, se suelen expresar mediante variaciones en la frecuencia de cada uno de los alelos de estudio.

Estos cambios genéticos que se producen a lo largo de las generaciones en una población son muy lentos, ya que dependen de la vida promedio de una especie. Es por esto por lo que el campo de la genética de poblaciones y la biología evolutiva en general han sido predominantemente cuantitativas, requiriendo de planteamientos teóricos, métodos numéricos y simulaciones por ordenador para su estudio (Servedio et al. 2014).

El estudio de la biología evolutiva, tanto académico como formativo, ha ido muy parejo con los avances tecnológicos por su propia naturaleza dilatada en el tiempo. Esto no ocurre con tanta relevancia en otros campos como puede ser la genética clásica o la biología celular, donde el abordaje experimental es mucho más viable por la inmediatez de las observaciones.

Además, diversos estudios muestran que la comprensión de los fundamentos teóricos quedan mejor integrados en el alumnado cuando se asimilan de forma activa (Drexel 2009). A modo general implica una interacción más directa con el contenido que se estudia más allá de una lectura repetitiva. Es bien sabido y está bien estudiado que plantear ejercicios o problemas durante una clase ayuda a afianzar los conceptos enseñados, esta es una forma básica, pero eficaz, de proporcionar un aprendizaje activo (Chen, Kalyuga, and Sweller 2015).

Este aprendizaje activo, enmarcado bajo la teoría filosófica del constructivismo (Nola and Irzik 2005), aboga que el conocimiento no se adquiere si no que se construye a través de la experiencia e interacción. Existen infinidad de formas distintas, plantear preguntas o incluso realizar experimentos sencillos son ejercicios que permiten al alumnado interactuar de alguna forma con la materia. A mayor nivel de interacción mejores son los resultados académicos.

Más relevante es este concepto de aprendizaje activo en la enseñanza de los conceptos de genética de poblaciones en las aulas; es posible plantear problemas pero difícilmente se puede plantear un experimento clásico en laboratorio para el estudio de los cambios genéticos en una población. Por esto se suelen utilizar programas de simulaciones y/o aproximaciones numéricas, ya que facilitan la comprensión de los fenómenos clave que tienen lugar en la evolución de las especies de forma casi instantánea, normalmente mediante la obtención de gráficos y tablas de frecuencias; varias de estas herramientas permiten simular y visualizr la evolución en un aula mediante el uso de un ordenador. No solo eso, sino que el avance en la capacidad de computación de los ordenadores personales permiten obtener simulaciones más realistas, en menor tiempo y de forma más intiutiva; así como estudiar fenómenos más complejos de forma relativamente sencilla (Carvajal-Rodriguez 2008).

El potencial de estas herramientas de forma pedagógica es inmenso; por una parte permite a un alumno realizar un experimento de forma virtual; por la otra el estudiante puede manipular las condiciones para obtener distintos resultados en formato gráfico y compararlos. La comparación de un resultado esperado frente al resultado obtenido es muy importante ya que permite a la persona deshacerse de falsas creencias o preconcepciones erróneas de una materia. Fuerza al estudiante a establecer una reflexión interna o externa, con el profesor u otros alumnos, acerca de los resultados

y las conclusiones. También puede promover la curiosidad casi científica de averiguar el porqué de ese resultado, todo esto mientras se aprende lo estudiado.

Populus, desarrollado por la universidad de Minnesota, es una de estas herramientas de modelización de procesos evolutivos. Su objetivo principal enseñar biología evolutiva en las aulas. Con este mismo fin, Populus se ha utilizado ampliamente por muchas universidades del mundo durante décadas debido a que es un programa muy sencillo de instalar desde su web, así como de utilizar. Consiste de una única interfaz gráfica con un pequeño menú donde seleccionar el tipo de "evento" evolutivo. Una vez seleccionado aparece una nueva ventana puedes ajustar algún parámetro del modelo; al final se obtiene el gráfico en una ventana adyacente, tal y como muestra la figura.

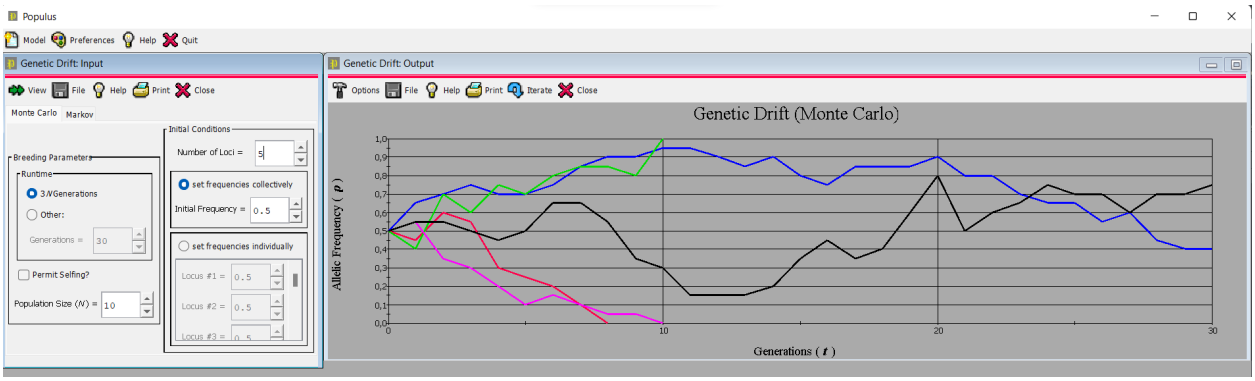


Figure 1: Interfaz gráfica de Populus para simular la deriva genética

Tanto Populus como otras herramientas similares supusieron una gran revolución a la hora de enseñar conceptos de biología ya que permitía una forma sencilla, novedosa y rápida de interactuar con la materia. Sin embargo, con el paso de los años, el programa no se ha adaptado ni a las necesidades educativas de los alumnos de hoy en día ni a los avances en el campo de la programación.

Populus es un programa que ya no supone una novedad en su uso, los alumnos están acostumbrados al manejo constante de programas y aplicaciones mucho más complejas y sofisticadas en su día a día. Su interfaz es extremadamente simple, esto hace que el usuario pueda obtener un gráfico cualquiera simplemente pulsando de dos a tres botones, sin comprender exactamente cuál es el proceso subyacente. Para asimilar conceptos es necesario que ocurra una interacción real entre el software y la persona, esta sencillez puede ser útil si lo que se busca es un producto o programa convencional, sin embargo en el ámbito de la enseñanza es muy superficial.

Si a un alumno se le plantea un problema que debe resolver con este programa el estudiante simplemente accederá a la aplicación y obtendrá el gráfico que se buscaba. Tal y como se ha comentado anteriormente, esto no estimula la curiosidad porque, por su simpleza, es muy complicado equivocarse a la hora de utilizarlo o plantear un problema que suponga un verdadero reto al alumno.

La interactividad con un programa es un paso necesario al hablar de aprendizaje, de la misma manera que ocurre en la vida real al aprender a usar herramientas. El usuario, alumno en este caso, primero debe familiarizarse con esa nueva herramienta para conocer lo que puede y no puede hacer; incluso hasta cómo funciona o por qué lo hace. Una vez aprehendido, el usuario podrá utilizarla de una mejor forma, y podrá interpretar mejor lo que está ocurriendo.

Cuando interactividad se reduce a la mínima expresión la curva de aprendizaje previo se reduce, esto lleva a que el usuario lleve a cabo el proceso con éxito, pero sin comprender el propio modelo,

sin explorar las limitaciones de éste e incluso perdiendo el interés en cómo se ha llevado a cabo. Esto resulta, en definitiva, en un aprendizaje incompleto y deficiente.

Este déficit en el aprendizaje es, además, muy difícil de encontrar y solventar hasta que ya es tarde. Al fin y al cabo, el alumno ha podido realizar la tarea o problema pedido con éxito y en tan solo unos minutos, es normal que el profesor asuma que los alumnos han comprendido hasta cierto punto el fundamento teórico que subyace al modelo, sin embargo esto no es así; este problema de aprendizaje se debe al propio programa por lo comentado anteriormente. La sencillez crea una falsa ilusión de comprensión, del mismo modo que ver la imagen obtenida en un microscopio no hace que comprendas lo que se ha observado, ni el propio funcionamiento del microscopio.

A todo esto se le suma otro factor: la importancia de la programación. La programación está tomando cada vez más relevancia en la sociedad debido a la digitalización de la mayoría de procesos, también en el mundo académico donde se utilizan con mucha frecuencia diversos lenguajes de programación para llevar a cabo tareas repetitivas, tediosas, complejas o específicas de un campo como puede ser el análisis de datos biológicos. A esto se le une la utilidad de la programación como herramienta de aprendizaje, conocer un lenguaje de programación permite analizar los problemas desde una perspectiva diferente, más analítica, incluso otra forma de pensar(Liao and Bright 1995).

Hace años suponía un gran obstáculo añadido, ya que lo que se buscaba era enseñar una materia en concreto, y muchas personas no tenían un ordenador en sus casas. En la actualidad, los alumnos pueden desenvolverse con facilidad en los ordenadores por lo que introducirlos en el mundo de la programación a través de programas de este estilo proporciona un doble beneficio. Por un lado, permite obtener un conocimiento que va más allá de la propia herramienta utilizada, por otro lado permite al alumnado aproximarse a un nuevo mundo como es el de la computación a través de conceptos familiares, en este caso la biología. Populus carece de todo esto al tener una interfaz gráfica sencilla. Ofrecer al alumnado de una interacción más directa con el software no tiene por qué suponer un problema a día de hoy y puede permitir que el propio usuario conozca y se interese por lo que ocurre tras el "botón" que se pulsa.

Por último, otro problema de Populus es su poca reproducibilidad. Tal y como se observa en la Figura 1, el resultado es un gráfico sobre el cual es muy complicado obtener conclusiones, analizar el procedimiento y, en definitiva, entender el experimento virtual que se ha llevado a cabo. Para los estudiantes de ciencias, comprender el procedimiento es igual o más importante que los resultados, por lo que no permite estudiar los pasos llevados a cabo ya que estos están ocultos bajo la interfaz. Ofrecer herramientas de simulación genética puede aumentar la reproducibilidad de una simulación(Adamack and Gruber 2014).

Más allá de Populus existen otros simuladores de evolución como simuPop (Peng and Kimmel 2005) o SLiM (Messer 2013), ambos programas surgen con una finalidad muy distinta a Populus aunque vienen a solucionar algunos de sus problemas. Sin embargo, ejecutar estos programas supone una barrera de entrada alta para usuarios inexpertos debido a que su funcionamiento requiere de ciertos conocimientos en programación, y para ello es necesario saber como instalar un intérprete/compilador, cómo y dónde descargar el programa, entre otros obstáculos que dificultan su uso.

La genética de poblaciones es un campo que puede aprovechar bien los avances técnicos para brindar una aprendizaje adecuado, contextualizado e interdisciplinar de sus conceptos.

Objetivos

Bajo las premisas de aprender sobre procesos evolutivos e introducirse en la programación para estudiantes de ciencias biológicas nace Populy, un paquete de código abierto desarrollado en Python de simulación de procesos evolutivos. El paquete, aunque sencillo de utilizar, requiere de conceptos de programación básicos, es por ello por lo que también se han creado unos materiales adicionales que servirán como complemento al programa, permitiendo una interactividad mayor con el software, un aprendizaje dinámico de los conceptos clave de la programación y de los procesos fundamentales de la evolución de las especies.

Los objetivos de Populy son, por tanto:

- Aprender sobre los procesos fundamentales de la genética de poblaciones.
- Ofrecer un aprendizaje interactivo, evitando el uso de botones, para que el usuario deba experimentar las distintas posibilidades de resolver un problema.
- Sencillez de uso. Debe tener una curva de aprendizaje sencilla pero que a su vez suponga un cierto esfuerzo intelectual.
- Introducir a los estudiantes de biología a la programación. No se pretende enseñar a programar pero sí familiarizarse con algunos de los conceptos clave.
- Flexibilidad. El programa debe permitir modificaciones y cambios con el tiempo.

Al tratarse de un programa de código abierto el software puede ser utilizado por cualquier persona. Además de esto, debido a su implementación, Populy puede adaptarse con el tiempo a los requerimientos de los alumnos, mejorando su funcionalidad e incluyendo nuevos tipos de simulaciones.

Mediante este paquete el usuario podrá invocar una o múltiples poblaciones, crear a sus individuos, hacerla evolucionar y obtener un *output* gráfico y tabular donde poder estudiar y comprender de forma más visual algunos de los mecanismos que se enseñan en las clases como son el equilibrio mutación-selección, la eficacia biológica en una población, la recombinación, la deriva genética, entre otros en menos de 10 lineas de código. Todo esto junto a una serie de explicaciones sobre el funcionamiento del paquete y de sus métodos clave, unos ejercicios o casos problema que podrán ser implementados o modificados en las aulas, permitiendo al estudiante un aprendizaje activo de los conceptos y una libertad mayor que la brindada por programas como Populus.

Computación como herramienta educativa

Alfabetización informática

La tecnología está cada vez más presente en una sociedad digitalizada, muchas de las interacciones sociales se llevan a cabo por medios virtuales, el usuario promedio pasa más de X horas interactuando con un dispositivo electrónico: transacciones bancarias,compra-venta de productos, consumo de nuevos contenidos como juegos y películas, etc son algunos de los muchos lugares donde la digitalización ha acaparado por completo el mundo tradicional.

Es innegable el impacto que estas herramientas están teniendo en la sociedad. La educación también debe adaptarse a estas nuevas tecnologías a un ritmo mayor incluso del que lo hace la sociedad(Iskrenovic-Momcilovic 2018). Al fin y al cabo, parte de la educación consiste en proporcionar al alumno unas herramientas para que se desenvuelva correctamente en el futuro. Por lo tanto, las competencias informáticas debe procurar ser vértebra del resto de contenidos.

No solo ésto, sino que estas herramientas tienen un gran potencial para mejorar o complementar la enseñanza ya tradicional. La inmensa cantidad de recursos web que están a disposición del alumnado ya de por sí facilita tareas como la búsqueda de información. El uso de diapositivas en clase puede complementarse con una actividad en grupo utilizando herramientas como Kahoot, que permite evaluar los conocimientos, llevar un registro actualizado de cada alumno, favorecer el trabajo en equipo, entre otros.

Las herramientas informáticas también facilitan el trabajo de los profesores. Hasta aplicaciones tan sencillas como Excel pueden actuar como una pequeña base de datos donde realizar cálculos sencillos de la nota.

A todo esto se le conoce como alfabetización informática: la capacidad de utilizar las tecnologías de la información y la comunicación para realizar tareas cotidianas. Se trata de un conjunto de competencias básicas que todos los ciudadanos deben tener para vivir y trabajar en un mundo cada vez más digitalizado.

La alfabetización informática es una habilidad esencial en la sociedad de hoy. A medida que la tecnología se vuelve cada vez más integral, es esencial que pueda ser utilizada por todos de manera efectiva. La alfabetización informática no solo se refiere a saber cómo usar una computadora, sino también a entender cómo funcionan los sistemas y aplicaciones informáticas.

En un mundo en el que la tecnología está cambiando constantemente, es vital que las habilidades informáticas se mantengan al día. La alfabetización informática permite estar mejor preparados para el futuro, ya que proporciona herramientas para adaptarnos a los cambios y aprender nuevas habilidades.

Tener un buen nivel de alfabetización informática proporciona mejores habilidades para desenvolverte en una sociedad cada vez más digitalizada y competitiva. Las personas con buenas habilidades informáticas tienden a tener mayores oportunidades de éxito en la vida(???), ya que están mejor preparadas para aprovechar las nuevas tecnologías (Liao and Bright 1995).

La alfabetización informática también es importante para la seguridad personal y la protección de la privacidad. A medida que más y más de nuestra vida se lleva a cabo en línea, es crucial que sepamos cómo proteger nuestros datos. Las personas con buenas habilidades informáticas son menos propensas a ser víctimas de fraude o robo de identidad, por ejemplo.

En resumen, la alfabetización informática es una habilidad esencial para el éxito en la sociedad de hoy a medida que la tecnología se vuelva cada vez más integral.

En cuanto a la educación, los estudiantes conviven con multitud de herramientas informáticas en su día a día; están habituados a una interacción digital mucho más profunda y como tal requieren de más recursos y herramientas de alfabetización informática. El objetivo educativo debe ser incrementar la alfabetización informática de los estudiantes para que estén mejor preparados para el mundo laboral y la sociedad en general.

Esta alfabetización debe estar presente en todas las etapas de la vida y en todos los niveles educativos, se trata de dar a la persona la habilidad de usar las nuevas herramientas para que pueda desenvolverse correctamente en su entorno. Así pues, aunque todas las personas deben tener una alfabetización mínima no será para todos la misma. La educación debe adaptarse a estas nuevas tecnologías que permiten realizar de forma más eficiente, dinámica e interactiva muchas tareas que anteriormente resultaban tediosas e incluso imposibles de realizar.

El avance en la educación y en la capacidad de abstracción de los alumnos se debe, en parte, a un aprendizaje y uso de las herramientas informáticas correspondientes ya que permiten ahorrar

tiempo y dedicar el esfuerzo a aquellas tareas que requieren de razonamiento, así como la presencia de nuevos estímulos y problemas.

El uso de las herramientas ofimáticas, internet e incluso calculadoras son competencias de la alfabetizacion informática. Sin embargo, estas herramientas son necesarias pero no suficientes para desenvolverse en la sociedad de hoy en día. Una de las nuevas tareas pendientes en la educación es el aprendizaje de la programación (Mishra et al. 2013).

La programación es una herramienta más, y como cualquier lenguaje natural requiere un cierto nivel de abstracción. Esta abstracción y "forma de ver el mundo" se le ha denominado pensamiento computacional Shute, Sun, and Asbell-Clarke (2017), siendo éste un subgrupo del denominado pensamiento analítico. La esencia de este modelo de pensamiento centrado en la abstracción y la automatización de procesos. De hecho, existe un debate público donde se le otorga el aprendizaje de la programación la misma importancia que aprender a escribir. La capacidad de abstracción es un buen indicador de la capacidad de resolución de problemas (Bergmann and Wilke 1996).

Esta herramienta, aunque similar a las anteriores, también es diferente en el sentido de que es mucho más abierta que el resto. Potencialmente permite hacer casi cualquier cosa que le puedas pedir a un ordenador, lo que abarca desde una sencilla suma hasta el uso de programas informáticos complejos. El usuario medio interactúa con las aplicaciones mediante el uso de interfaces gráficas, estas interfaces facilitan el uso pero restringen las posibilidades de muchos programas y servicios web, esto es, no permiten explotar el potencial. Al introducir al alumnado a la programación se le abre un nuevo campo de conocimiento donde poder utilizar programas menos "intuitivos" pero mucho más potentes, en virtualmente cualquier ámbito, incluido la biología.

No solo eso, sino que la demanda social de un conocimiento de programación está en auge. Una gran cantidad de trabajos requieren de unos conocimientos básicos de programación: estructuras y tipos de datos, bases de datos o análisis de información. Proporcionar a los alumnos unas competencias mínimas puede repercutir favorablemente en su integración al mundo laboral.

La capacidad de indicar a una máquina las instrucciones para ejecutar una acción ha existido desde hace siglos; ya en el siglo IX se describe una máquina que toca la flauta de forma automática (Koetsier 2001), sin embargo el primer programa surge en 1843 por Ada Lovelace desarrollado para la máquina analítica que nunca llegó a construirse. La programación va ligada a la creación y uso de las máquinas para instruirlas. Actualmente el objetivo es el mismo que hace cien años, comunicarse con una máquina para que lleve a cabo determinadas operaciones, es una forma de dar instrucciones que un ordenador pueda traducir a su propio lenguaje, para ejecutar una tarea. Con la ubicuidad de los ordenadores en los hogares y las aulas y la creación de multitud de lenguajes de programación de alto nivel y con sintaxis similar a la lengua humana surge la posibilidad y la necesidad de aprender a programar.

Sin embargo, la programación ha evolucionado: existen multitud de lenguajes de programación diferentes con propósitos diferentes. También los recursos para aprender en la web son enormes.

Lenguajes de programación

Un lenguaje de programación es un un medio de comunicación con una computadora, se constituye de una serie de instrucciones (variables, funciones...) con estructura fuertemente definida que proporciona las herramientas necesarias para llevar a cabo ciertas tareas.

Existen multitud de lenguajes de programación, que se clasifican históricamente según dos criterios:
1. Nivel de abstracción: esto es, según la similitud al lenguaje empleado por los ordenadores,

también conocido como lenguaje máquina. Se le denomina de alto nivel a un lenguaje con una sintaxis más abstracta, es decir, similar al lenguaje humano y menos dependiente del *hardware* de la máquina mientras que uno de bajo nivel es más similar al lenguaje máquina pero más dependiente. 2. Según el paradigma: aunque la mayoría de lenguajes de programación permiten llevar a cabo las mismas tareas e implementar los mismos algoritmos no todos lo hacen de la misma manera, existen diferentes formas de estructurar el código, los paradigmas más conocidos son Orientado a Objetos, funcional y multiparadigma. 3. Según su propósito: Aquellos con una utilidad muy restringida, pensados para una tarea concreta se conocen como lenguajes de propósito específico mientras que el resto son de propósito general.

A la hora de introducir a estudiantes a uno u otro lenguaje surgen diversas cuestiones: ¿se debe enseñar primero lo más complejo, para después utilizar lo sencillo? ¿cuál tiene más aplicaciones relacionadas con el campo de estudio? ¿con qué finalidad se va a utilizar en el aula?. Las respuestas son muy diversas, en muchas aulas se comienza enseñando C o C++, ambos lenguajes son de bajo nivel pero muy eficientes. Se suele elegir este tipo de lenguajes en carreras o profesiones que requieren un conocimiento exhaustivo de la computación y sus estructuras (Johnson and McQuistin 2020). También se defienden estos lenguajes no tanto por su utilidad sino porque una vez entendido su funcionamiento es más sencillo extrapolar los conocimientos a otros lenguajes.

Sin embargo, en los últimos años está cobrando relevancia la enseñanza de lenguajes más sencillos para un usuario inexperto. Debido a su curva de aprendizaje hacen que estos lenguajes permitan obtener resultados fácilmente, aumentan la versatilidad y tienen una gran comunidad detrás.

De este tipo de lenguajes existen dos que son ampliamente usados en el ámbito académico: Python y R.

Para el propósito del presente trabajo ambos eran igualmente útiles ya que son de alto nivel y multipropósito, tienen sintaxis similares y tipado dinámico. En definitiva, son lenguajes sencillos de aprender.

La instalación es sencilla, poseen una enorme cantidad de recursos para aprender gratis de forma autodidacta y la sintaxis es intuitiva. Todo esto hace que un alumno sin conocimientos de programación puede programar y entender las primeras lineas de código en menos de 15 minutos.

Para la experiencia de usuario las diferencias son minúsculas, en todo caso R es un lenguaje muy usado en el ámbito académico, en especial para realizar análisis y tratamiento de datos. Por otra parte, Python es más utilizado en el ámbito profesional y la industria, con múltiples enfoques aunque destaca por sus aplicaciones en aprendizaje máquina.

No obstante, a la hora de desarrollar el paquete Populy sí se presentaron algunas diferencias:

- R no es estrictamente multi-propósito. Este lenguaje fue creado por y para estadistas(R Core Team 2020), su paradigma es principalmente el funcional: creación y uso de funciones de manera similar al comportamiento de las funciones matemáticas. Python sí es multi-propósito y soporta de forma adecuada la orientación a objetos, que se explicará más adelante.
- Soporte con libretas *Jupyter*, como se menciona en el siguiente capítulo. Estas libretas fueron creadas para Python y, aunque a día de hoy soportan R y otros lenguajes es mucho más común encontrarse utilizando estas libretas en Python antes que en R.

Como se puede observar, las diferencias son muy sutiles y cualquiera de ambos podría haberse empleado. La finalidad al escoger uno u otro lenguaje era emplear un medio que no fuese complejo y que permitiera a un usuario inexperto introducirse en el mundo de la programación.

Python

Python es un lenguaje de programación de alto nivel multi-paradigma de propósito general, fue diseñado por Guido Van Rossum en 1991 (Van Rossum and Drake 2009). El objetivo de este lenguaje es facilitar la interpretación del código, su aprendizaje y su uso.

La facilidad aprendizaje y la gran comunidad que tiene a su alrededor son algunos de los motivos por el cual Python es el cuarto lenguaje más popular y con mayor proyección ("Stack Overflow Developer Survey 2021 -Popular Technologies," n.d.).

Aunque Populy podría se desarrolló en Python principalmente debido a su sencillez de uso por parte del usuario (Johnson and McQuistin 2020; Jayal et al. 2011), sin embargo la presencia de la gran comunidad colaborativa, el gran número de herramientas y mejor paradigma de orientación a objetos que R fueron un añadido al optar por este lenguaje. El paradigma de programación establece que gran parte del código, sino todo, debe estar encapsulado en Objetos, estas entidades deben tener un propósito común, es decir, cada objeto creado debe tener una características que lo definan ("Object-Oriented Programming" 2022). La interacción entre el código debe ser una interacción entre objetos siempre que sea posible.

Los objetos son una herramienta que permiten encapsular el código similar en bloques comunes, haciendo el programa más legible. Esto es lo que se buscaba a la hora de desarrollar Populy debido a que uno de los objetivos es crear una población que contendría unos individuos, mediante el paradigma de la orientación a objetos se entiende que tanto población como individuo deben ser dos clases de objetos distintas, que tendrán un tipo de relación determinada que les permitirá interactuar.

Jupyter notebook

El proyecto Jupyter es una colaboración para el desarrollo de herramientas de libre uso (open-source) para realizar análisis exploratorios y computacionales de forma interactiva. El proyecto Jupyter fue creado en 2014 a partir de Ipython aunque actualmente soporta más de 100 lenguajes de programación diferentes (Kluyver et al. 2016). Dentro del ecosistema Jupyter se encuentran las *Jupyter notebooks* o libretas Jupyter, estas libretas son un entorno web que tuvo su origen en la facilidad de realizar y compartir análisis de datos. Sin embargo desde hace unos años esta herramienta se está empleando en las aulas con propósitos educativos de una gran variedad de formas distintas.

La estructura externa de una libreta Jupyter es muy simple ya que consta de bloques de información, estos bloques pueden ser de texto, escrito en formato Markdown, o bloques de código tal y como se puede observar en la figura adjunta. Las posibilidades que surgen de combinar estos dos elementos son varias, desde explicar de forma detallada un bloque de código, sin tener que recurrir a manuales externos o a libros, hasta la posibilidad de plantear problemas *in situ*. Por otra parte, los bloques de código no son estáticos, el usuario puede modificarlos, ejecutarlos y observar los resultados reales en el momento, así como probar a cambiar ciertos parámetros del bloque o poder crear su propio código, todo en tiempo real y de una forma mucho más dinámica que la que ofrecen los métodos tradicionales, ya sea en un formato word/pdf estático, donde el usuario no puede interactuar con los resultados, o una interfaz gráfica de usuario como puede ser Populus, donde la estructura programática queda oculta a la persona.

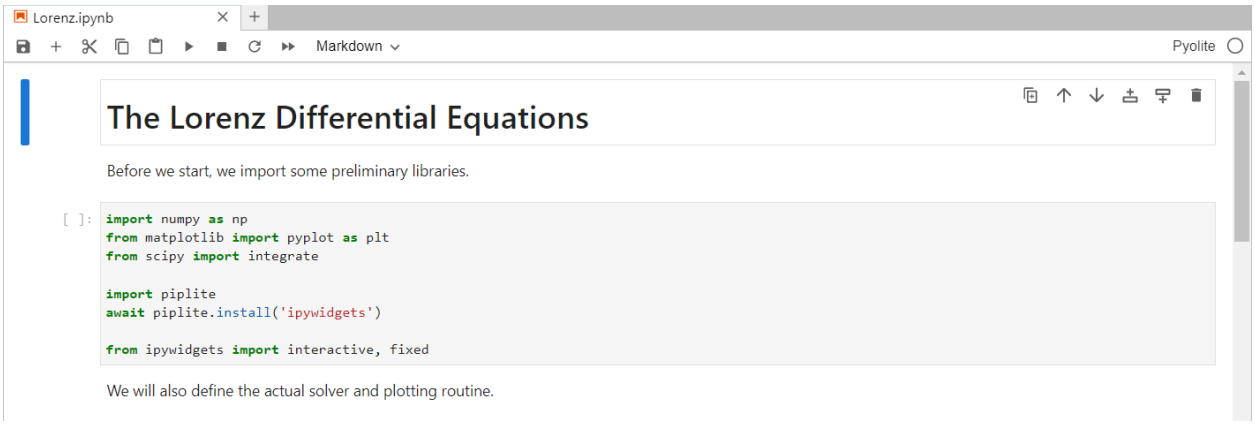


Figure 2: Estructura de una Jupyter Notebook de la documentación de (Jupyter.org)

Una libreta Jupyter es, por tanto, un documento interactivo, orientado a la computación, que permite crear una narrativa para explicar, analizar o enseñar conceptos.

Métodos de uso libreta Jupyter en el aula

Estas libretas se pueden utilizar igual que una presentación de PowerPoint, el profesor puede crear o utilizar una nueva libreta y en la misma clase quedará accesible a los alumnos para que éstos la ejecuten, al tiempo que se realiza una explicación más detallada de los conceptos.

Este método de uso de las libretas se conoce como se conoce como *Shift-Enter* (*Teaching and Learning with Jupyter* 2022), que es el comando utilizado para ejecutar un bloque de código. El alumno únicamente deberá leer el texto y a continuación ejecutar el código. Esta forma de usarlo puede ser útil para introducir al alumnado a este tipo de libretas aunque no explota todo el potencial que tienen las libretas Jupyter para la enseñanza.

El método más acorde con la filosofía de las libretas Jupyter es el de poder modificar el código y probar cosas nuevas, experimentar los cambios que se producen e interpretar tus propios resultados. Esto se puede lograr de diversas formas, la manera que se ha seguido a la hora de desarrollar las libretas para Populy es la del ejemplo trabajado o worked example (Sweller and Cooper 1985). Este método consiste en la introducción de un concepto teórico, junto al código necesario y más básico para llevarlo a cabo de forma guiada.Tras esto, el estudiante deberá realizar pequeñas modificaciones al código o crear un bloque de código propio y finalmente observar el cambio en los resultados o los errores que surjan durante el proceso. Por último, cuando el concepto teórico que se explique y el código que lo lleva a cabo queden claros se procederá a realizar un ejercicio desde 0, siguiendo los pasos explicados en la libreta anterior.

Este método puede suponer un reto, en especial la parte del problema, sin embargo este desafío intelectual junto a la interactividad de las notebooks fuerza al que estudiante a reflexionar sobre el problema o concepto teórico de forma activa. Demandar la resolución de una tarea de complejidad variable para obtener un resultado, ya sea correcto o incorrecto es una de las técnicas más eficaces para asentar un conocimiento.

¿Cómo utilizar una libreta Jupyter?

Las libretas Jupyter son documentos en formato json, este formato es particular y como tal requiere de herramientas específicas para su ejecución, no es posible abrir una libreta Jupyter con un procesador de textos convencional. Es por ello por lo que existen herramientas que pueden leer la información de las libretas de forma adecuada.

En local:

Para la ejecución de una libreta Jupyter en local se requiere una distribución de software "programa" que incluya Ipython. La distribución Anaconda es la más utilizada, Anaconda es una distribución de Python y R ("Anaconda Software Distribution" 2020). Al instalar Anaconda se instala el software necesario para utilizar las libretas Jupyter entre otras cosas. Tras su instalación simplemente consiste en abrir Jupyter desde Anaconda, tal y como se muestra en la figura, se abrirá una ventana en un navegador web y se podrá acceder al entorno Jupyter de forma sencilla.

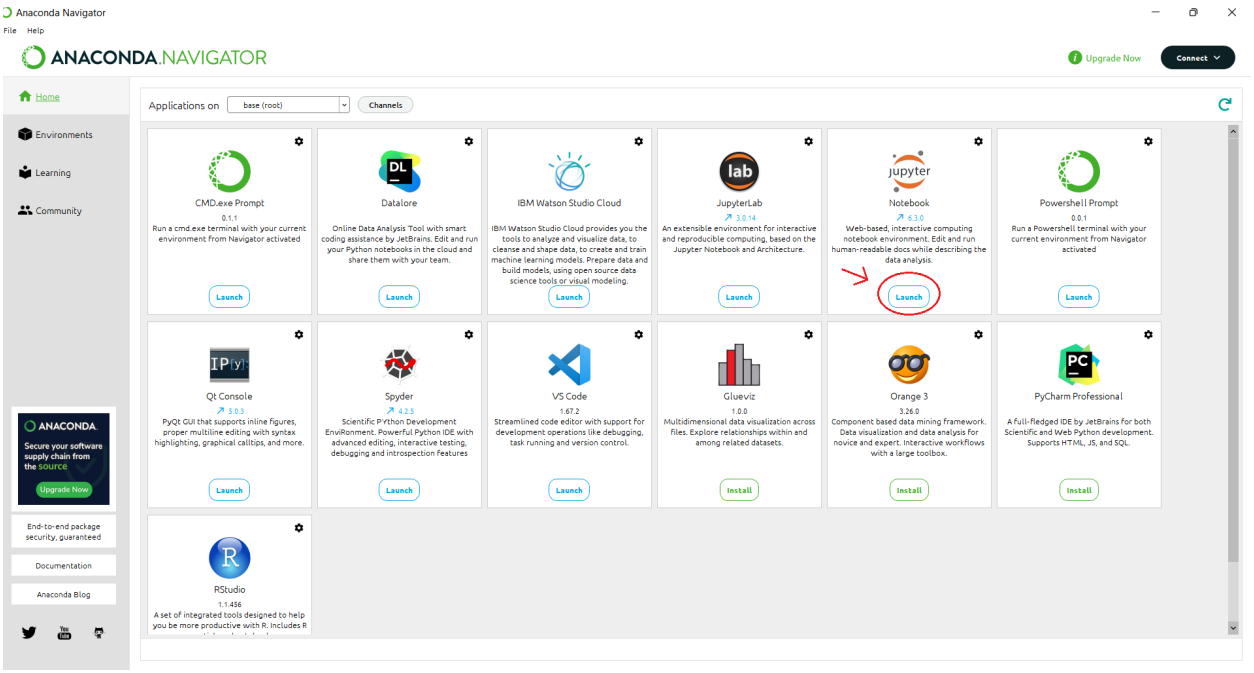


Figure 3: Navegador anaconda, en rojo se muestra la forma de abrir el entorno Jupyter

Esta forma es recomendada para utilizarlo de forma autónoma sin depender de servidores, su instalación no es excesivamente compleja y solo requiere llevarla a cabo una única vez. Además, Anaconda instala también Python y R por lo que podrá crear y ejecutar código en cualquier editor de texto avanzado (como VSCode, Spyder o Pycharm), y los usuarios más avanzados podrán crear entornos virtuales.

En remoto:

Una de las ventajas Jupyter *Notebooks* es que no requieren de forma obligatoria el proceso de instalación de un compilador o intérprete en local, de un editor de textos avanzado o de un IDE (Entorno de Desarrollo Integrado), únicamente es necesario una conexión a una libreta un servidor remoto.

Esta conexión a un servidor remoto se puede hacer mediante MyBinder (Jupyter et al. 2018). El estudiante simplemente deberá abrir una notebook ya creada que se encuentre en mybinder y podrá utilizarla. Si el uso de estas libretas no va a ser frecuente el trabajo mediante mybinder es una buena forma de introducir a usuarios inexpertos a la programación o para aulas donde Jupyter no sea una herramienta común.

Otro recurso web muy utilizado es Google Colab, la forma de uso es muy sencilla ya que simplemente requiere una cuenta de Google y la instalación de una extensión de navegador para crear o abrir las libretas.

Dado que el software está pensado para el uso en aulas se ha creado un repositorio en MyBinder donde se incluyen las libretas. La forma de acceder es tan simple como accediendo al enlace.

Genética de poblaciones

La genética de poblaciones es definida como una rama de la genética encargada de estudiar y describir los aspectos heredables de una población a través del tiempo (Okasha 2016). Este campo de estudio se vale de diversos aspectos de la biología evolutiva, en concreto de la síntesis evolutiva moderna, y de la genética cuantitativa para estudiar las diferencias genéticas de las poblaciones.

Cuando se habla de población e individuo no se hace referencia únicamente al concepto más extendido de población; puede ser cualquier grupo de individuos que comparten una serie de características capaces de generar descendencia. Bajo este paraguas existen poblaciones de animales, bacterias o incluso células de un organismo. Existen autores como Dawkins que hablan de población y selección de genes como explicación última de varios procesos evolutivos (Dawkins 1989). De hecho, diversos tipos de simulaciones se pueden entender mejor como una población de gametos portadores de un genotipo que dan lugar a otro gameto. Pese a ser una reducción y simplificación en el caso de poblaciones como la humana, estos modelos y simulaciones tienen cabida para explicar varios fenómenos evolutivos, con aproximaciones cercanas a las observadas en poblaciones reales.

Así pues, la amplitud de este campo de estudio es tanta como el número de poblaciones diferentes que existen, aunque compartan diversos atributos entre ellas, cada población es única y tiene sus particularidades asociadas a sus propiedades internas (estructura genética y reproductiva) como externas (ambiente y época). Ésto, unido a los largos tiempos asociados al cambio evolutivo, hacen que el abordaje experimental sea costoso en la genética de poblaciones (Servedio et al. 2014).

A modo de norma, una población se observa en un tiempo determinado X, en el cual se pueden medir sus propiedades e inferir otras. Dependiendo del tipo de aproximación, se puede intentar extrapolar las condiciones del tiempo X a una población a tiempo 0 o por el contrario hacer avanzar la población hasta un tiempo X+n, siendo n un número de generaciones.

Existen numerosas aproximaciones a este tipo de problemas, normalmente tratadas mediante análisis numérico. En los últimos años han cobrado popularidad las simulaciones de población porque permiten, como su nombre indica, simular todo el proceso evolutivo en una u otra dirección, obteniendo no solo el resultado final si no información del proceso (Carvajal-Rodriguez 2008).

A la primera forma de evolucionar una población (`X -> X+n`)se la conoce como *forward simulation* o simulaciones de tiempo hacia adelante (Peng, Kimmel, and Amos 2010). En estas simulaciones la población parte de un estado X en un momento 0 y se hace avanzar en el tiempo (generaciones) hasta llegar a otro estado. Durante el proceso se aplica una serie de funciones y/o modelos que permitirán una evolución de la población. Una vez terminada, se obtiene un informe en formato

gráfico o tabular donde queda constancia del proceso, del estado inicial y del estado final. Este tipo de simulaciones son más complejas computacionalmente ya que requiere crear diversos individuos durante varias generaciones; sin embargo, tanto el proceso como sus resultados son más similares a la evolución de las especies.

Existen otros métodos, como los basados en la teoría de la coalescencia (Nordborg 2000), que parten de una población final y buscan los antepasados más probables para ese estado de forma sucesiva hasta llegar al estado inicial(`X -> 0`). A estas simulaciones se las conoce como *reverse simulation* o simulaciones tiempo hacia atrás ya que su forma de funcionar es inversa. Son computacionalmente más eficientes y útiles cuando el objetivo es obtener un ancestro común para una población, sin embargo son menos realistas y menos intuitivas.

Para este trabajo se ha optado por las simulaciones de tiempo hacia adelante debido a que uno de los objetivos de Populy es permitir a los alumnos el aprendizaje de conceptos de biología evolutiva y genética de poblaciones, este tipo de simulaciones son muy sencillas de comprender porque emulan el comportamiento de una población y también son más útiles para explicar conceptos clave de la evolución.

Para comprender cómo llevar a cabo una simulación de este tipo primero es necesario conocer los aspectos fundamentales de genética de poblaciones y biología evolutiva que la definen.

Estructura poblacional

La definición de población está estrechamente ligado al concepto de especie. Este término es todavía más complejo de definir y acotar ya que es difícil establecer límites entre especies cercanas, por ello existen diferentes definiciones para el concepto de especie, cada uno con sus particularidades (Masters and Spencer 1989). *Grosso modo*, una población es conjunto de individuos de la misma especie que coexisten, que comparten una serie de características y que son capaces de generar nuevos individuos viables a nivel reproductivo(Soler 2002).

En el caso de las simulaciones genéticas, la población está definida por una serie de atributos fundamentales:

1. Individuos: unidad fundamental de la población
2. Tamaño: número de individuos de la población
3. Ploidia: Indica número de cromosomas homólogos, generalmente diploide o haploide.
4. Genoma: puede dividirse o no en cromosomas, generalmente indicado como número de genes y número de alelos por gen.
5. Tipo de reproducción: sexual o asexual.

Solo con estas características es posible crear el modelo más simple, conocido como modelo de Wright-Fisher, este modelo asume una serie de condiciones en la población que para un solo locus son:

- Sin selección: no actúa la selección natural sobre ningún alelo estudiado
- Sin mutación: no existe mutación en los alelos estudiados
- Sin migración: los individuos no desaparecen de la población ni se introducen nuevos
- Discretización de las poblaciones: no existen edades, generaciones no solapantes.

- Emparejamiento aleatorio: no existe selección sexual, los individuos se generan a partir del muestreo aleatorio de dos individuos de la población parental.

En el caso de varios loci se debe asumir independencia estadística entre ellos, por lo que su recombinación debe ser de `0.5`.

En definitiva, el modelo de Wright-Fisher para un locus estudia la evolución de las frecuencias alélicas tras un muestreo binomial de los gametos tal y como se muestra en la siguiente imagen

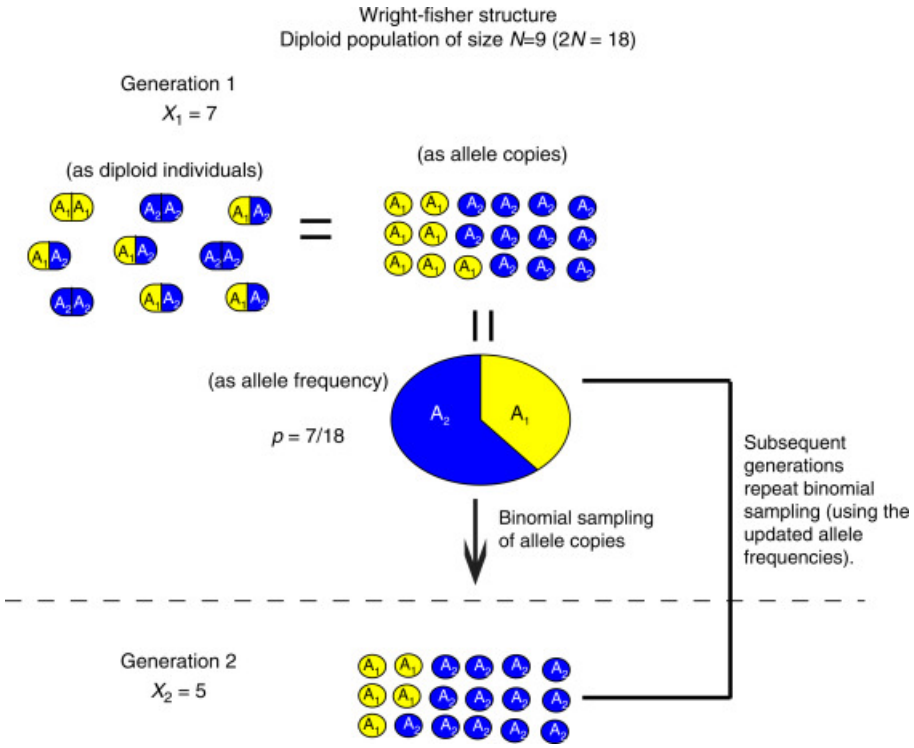


Figure 4: Cambio en las frecuencias alélicas de una población de Wright-Fisher diploide (Obtenido de Muirhead 2016)

Aun con este modelo simple se puede observar un cambio en las frecuencias alélicas tras el paso de las generaciones, esto se debe a la presencia de al menos una fuerza evolutiva.

Evolución y fuerzas evolutivas

La evolución es el proceso de cambio de las especies a lo largo de las generaciones, el estudio de la evolución se hace a nivel de la población y se suele expresar como el cambio en las frecuencias alélicas en el tiempo. La frecuencia alélica de una población es la frecuencia de cada alelo, es decir, cada variante de un gen, en una población. En una población diploide cada individuo podrá tener hasta 2 alelos diferentes.

En el contexto de simulación de evolución el concepto es más simple; se dice que una población evoluciona cuando se suceden las diferentes generaciones. Si se hace evolucionar una población del tipo de Wright-Fisher podría esperarse que las frecuencias alélicas de la población se mantuviesen constantes.

Darwin propuso en su libro el *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life* (Darwin 1859) un único modo de cambio evolutivo: selección natural. Tras los avances en genética y otros campos de la biología se desarrolló la nueva síntesis evolutiva (Carroll 2000). En ella, la selección natural es una de las cuatro fuerzas principales que provocan el cambio evolutivo.

Mutación

La mutación es la primera fuerza evolutiva, consiste en la aparición espontánea o inducida de cambios en el material genético. Estos cambios pueden tener orígenes distintos, no obstante la consecuencia última es la generación de diversidad, esto se debe a que propicia aparición de nuevos alelos o el cambio de un alelo a otro en la población. Esta fuerza, junto a la migración, son las únicas que permite introducir nuevos alelos en el conjunto de genes y alelos en una población, llamado **gene pool** y junto a la recombinación genética son las únicas fuentes de variabilidad genética (Soler 2002).

Al contrario que otras fuerzas esta ocurre a nivel de genes. Sin embargo, no todas las mutaciones son objeto de la selección; por lo general solo aquellas que tengan un efecto en el fenotipo y que sean transmisibles a la siguiente generación podrán ser sometidas a la selección natural.

Las mutaciones se pueden clasificar por su efecto en la población; existen mutaciones beneficiosas, perjudiciales o neutras.La inmensa mayoría de mutaciones, especialmente las de sitio único e indels tienen un efecto neutro. En el contexto evolutivo, se dirá que una mutación es beneficiosa cuando aumenta la eficacia biológica del individuo (en esa población) mientras que una perjudicial la disminuye. Por poner un ejemplo, una mutación que cause una disminución en la supervivencia de un individuo pero que aumente su índice reproductivo podría ser una mutación beneficiosa, a pesar de suponer un perjuicio al individuo.

Aunque los tipos de mutaciones son variados (SNPs,indels,cromosómicas. . .), para la simulación de evolución se tomará el el modelo más simple de mutación: una mutación es el cambio de un alelo mayor al menor `P -> q`. Las mutaciones son transmisibles, por lo que su punto de origen en caso de ser un individuo real ocurriría en los gametos de los padres. Ademas, la frecuencia de mutación será independiente para cada locus.

Selección natural

Junto a la mutación, la selección natural es la fuerza evolutiva más conocida. Fue el primer mecanismo de cambio evolutivo descubierto(Darwin 1859) y se basa en los siguientes principios:

1. Los individuos son diferentes entre ellos (variación)
2. Parte de esa variación es transmisible generacionalmente (herencia)
3. Algunos individuos sobreviven y se reproducen más que otros
4. El éxito de supervivencia y reproducción no es aleatoria, sino debida a las diferencias entre los individuos.

Bajo estos 4 principios, el mecanismo de selección natural se comprende como la eficacia diferencial de un genotipo en una población en un ambiente determinado, también llamado evolución adaptativa. Es decir, aquellos individuos con unas características genéticas (variación) heredables mejor

adaptadas a un ambiente sobrevivirán y darán más descendencia en promedio, haciendo que esas características genéticas se vean más representadas en la generación siguiente.

La selección natural debe entenderse como una fuerza que actúa sobre los genes de un individuo pero que solamente se ve reflejado en una población, más concretamente en sus frecuencias alélicas.

Además, la selección actúa sobre el fenotipo pero solo aquellos rasgos transmisibles serán sometidos a selección natural. El mecanismo principal es conocido como *fitness* o aptitud biológica, que refleja el éxito reproductivo de unos alelos sobre otros.

Esta fuerza, por lo general, es la que provoca un mayor cambio en las frecuencias alélicas de la población. Así como la mutación permite obtener nuevos alelos, la selección actúa haciendo que se vean favorecidos o perjudicados, según el cambio que provoquen.

Por tanto, para simular el proceso de selección natural se requiere una población que cumpla los postulados mencionados anteriormente:

1. individuos variables: para conseguir esto solo es necesario que la población inicial (generación 0) tenga alelos diferentes.
2. variación heredable: la características genéticas de un individuo deben ser heredables, esta transmisión de las características genéticas viene determinada por la arquitectura genética.

3 y 4. Supervivencia/éxito reproductivo diferencial: es posible simular la aptitud biológica de dos formas: asignando un valor que indique el numero promedio de descendientes según su genotipo o un valor que indique su tasa de supervivencia. Para esta simulación el *fitness* de un individuo según una probabilidad de supervivencia para cada genotipo o alelo, es decir , que la variación genética resulte en diferente éxito reproductivo. A modo de ejemplo: aquellos individuos con un genotipo cuyo *fitness* sea `0.2`tendrán un 20% de probabilidades de supervivencia.

Migración

La migración se puede entender como el intercambio genético (*gene flow*) entre dos poblaciones, para que se lleve a cabo estas dos poblaciones han debido estar separadas por alguna barrera física, temporal, comportamental o reproductiva hasta un punto determinado donde esa barrera desaparece e intercambian material genético de forma puntual o periódica. Este intercambio puede generar nuevos alelos en la población o incrementar la presencia de uno minoritario.

Para modelar la migración de forma realista se requiere de otra población y romper el aislamiento durante el proceso de evolución. En la simulación no se encuentra implementada la migración.

Deriva genética

En ausencia de mutación, selección, migración y suponiendo un apareamiento aleatorio se hace evidente un mecanismo llamado deriva genética.

Para entender este mecanismo es necesario ver a cada generación no como una copia sino como una muestra, una porción, de la generación previa. En este sentido, la generación hija es el resultado de tomar ciertos individuos de la generación parental. Al estudiar la población padre se observarán

unas frecuencias alélicas que no serán exactamente iguales a las de la población hija, simplemente por el efecto de hacer un muestreo aleatorio en una población finita.

Al cambiar las frecuencias alélicas ya existe un cambio evolutivo y por tanto se dice que la población ha evolucionado, aunque sea por el azar.

Una consecuencia habitual de la deriva genética es la pérdida de alelos, lo que conlleva una disminución de la variabilidad genética y una pérdida de la heterocigosidad (Muirhead 2016), esto es, de los individuos heterocigotos.

Este mecanismo está presente en todas las poblaciones con reproducción sexual aunque el efecto es despreciable para poblaciones grandes ya que el error en el muestreo (en la selección de padres) será prácticamente despreciable. Sin embargo, al reducir el número de individuos en una población es posible observar sus efectos.

Para simular la deriva genética, es conveniente reducir el tamaño de la población a 10-50 individuos, eliminar el resto de fuerzas (selección y mutación) e incluir varios loci en la población para ver la pérdida de alelos que se produce. Tras las suficientes generaciones, todos los alelos quedarán fijados.

Equilibrio de Hardy-Weimberg

Cuando no actúa ninguna de las cuatro fuerzas mencionadas se puede alcanzar una situación de equilibrio llamado Hardy-Weimberg. En este estado se dice que, tras una generación de apareamientos aleatorios, las frecuencias alélicas de la generación padre serán iguales a las de la generación hija. Una población en equilibrio se trata de una población sin cambio evolutivo. Aunque esto no ocurre en poblaciones reales, simular una población en equilibrio de Hardy-Weimberg ya permite estudiar diversas características de una población y permite compararlas con otros casos donde actúe una o más fuerzas. Un modelo de Wright-Fisher con un tamaño de población infinito cumplirá las condiciones para que se de el equilibrio de Hardy-Weimberg.

Tener una población en equilibrio permite, por un lado, realizar una simulación de forma sencilla y por otro, tener una población de "referencia" con la que comparar las medidas. Asumiendo una población en equilibrio se puede realizar un test estadístico para comprobar frecuencias alélicas esperadas vs observadas y analizar si la simulación se desvía de forma significativa de lo esperado, en cuyo caso se dirá que la población se encuentra sometida a cambio evolutivo.

Con todo esto, es más habitual simular las poblaciones sometidas a una o más fuerzas evolutivas para estudiar sus efectos. Es posible incluir tanto una frecuencia de mutación para cada locus; unos valores de eficacia biológica, o índice de supervivencia, para cada genotipo o para cada alelo y cambiar el tamaño de las poblaciones para observar los efectos de la deriva genética.

Estas son las características generales de la población, sin embargo existen otros parámetros que corresponden al individuo, de todos ellos el más relevante es su estructura o arquitectura genética.

Arquitectura genética

En una población diploide cada individuo posee dos cromosomas homólogos, en éstos se sitúa todo el conjunto de loci del individuo. Se utiliza la notación de loci en lugar de genes porque es más correcta ya que un locus hace referencia a un lugar del cromosoma que no tiene por qué ser un gen. Sin embargo, para los propósitos explicativos, se puede utilizar genes para mayor claridad.

Por defecto, cada individuo tendrá un par de genes, cada uno con dos alelos posibles `A/a y B/b` agrupados en un único par de cromosomas homólogos, esto constituye el genotipo del individuo.

Al tratarse de una población con reproducción sexual, los individuos también tendrán un sexo que podrá ser macho o hembra, la determinación del sexo viene dada por los cromosomas sexuales que podrá ser XY,ZW o X0, de forma similar a cómo se determina, simplificando, el sexo en mamíferos, aves y algunos insectos.

Así pues, un individuo está constituido por su autosoma, sus cromosomas sexuales, unos padres y un identificador único, que incluye su número de individuo y la generación a la que pertenecen.

Existen otras dos características de la población que intervienen durante el proceso de creación y evolución únicamente cuando se estudian dos o más genes. Cuando se genera la generación 0, esto es, la generación inicial, puede intervenir un factor llamado desequilibrio de ligamiento, este valor indica la probabilidad de que dos genes segregen independientemente. Así pues a un menor valor de este número indicará una mayor asociación estadística de los alelos.

Durante el proceso evolutivo es el valor de la frecuencia de recombinación el que representa la independencia estadística. En términos biológicos, este valor numérico indica la frecuencia con la que dos cromosomas homólogos intercambian su material genético durante el proceso de recombinación que tiene lugar en la meiosis. Este factor es similar al desequilibrio de ligamiento en cuanto a su implementación pero en este caso indica cuál es la probabilidad de que a partir de los gametos parentales se generen gametos recombinantes.

Así pues, el desequilibrio de ligamiento se utiliza para crear una población mientras que la frecuencia de recombinación se usa durante la evolución.

Populy: uso y estructura

Métodos de uso

Tras el desarrollo del paquete, se han creado unos *notebooks* para mostrar cuáles son los métodos esenciales para llevar a cabo la creación, iniciación, evolución y obtención de resultados de una población, así como algunos ejercicios para que el usuario pueda experimentar y crear diferentes poblaciones. Como se ha comentado anteriormente, Populy está pensado para su uso en un *notebook*, aunque se puede utilizar de forma local con un editor de texto como *visual studio code* o un IDE como *Pycharm* o Rstudio. A continuación, se procederá a explicar el proceso que se debe seguir para el uso correcto del paquete desde 0:

Lo primero es necesario instalar, ya sea en local o remoto, el paquete Populy. El proceso de instalación de paquetes en una *jupyter notebook* de Python es muy sencillo, únicamente es necesario ejecutar la siguiente linea de código.

```
!pip install Populy
```

En el caso de trabajar para el caso de trabajo en local, si el usuario se encuentra en Windows primero deberá instalar el compilador de Python que se encuentra en su página oficial.

En el caso de dispositivos macOS o Linux, estos ya vienen por defecto con Python instalado, sin embargo su versión es la 2.0 por lo que es necesario actualizar a la versión más reciente que también se encuentra en su página web.

Tras esto deberá abrir la ventana de comandos de windows (cmd) o powershell e instalar el paquete mediante la siguiente línea de código:

```
pip install Populy
```

Tras esto, puede abrirse el terminal y ejecutar el programa; es altamente recomendable utilizar un editor de textos avanzado como Visual Studio Code o un entorno de desarrollo como Pycharm para poder crear y ejecutar scripts de forma sencilla.

Una vez instalado el paquete es necesario cargarlo en memoria (entorno de ejecución) para que pueda ser accesible de forma temporal, a este proceso se le conoce como carga o importación de librerías y paquetes. En Python, simplemente es necesario ejecutar la siguiente línea:

```
import Populy
```

Esta es la forma sencilla de cargar el paquete en el entorno de ejecución, sin embargo será preferible realizarlo de esta manera. Para entender la sintaxis de los *imports* se recomienda consultar el estructura.

```
from populy.population import Population
```

En definitiva, con este tipo de importación se selecciona qué archivo se va a utilizar de todo el paquete y facilita el manejo de sus objetos.

Una vez completado el proceso de instalación ya es posible crear una población: el diagrama de uso habitual del paquete queda reflejado en el siguiente esquema:

Así pues, el primer paso es la creación de una población. Este proceso también se le conoce como instanciación y se lleva a cabo con la siguiente linea de código:

```
# creación de una población
poblacion = Population()
```

En este paso se ha creado una población, como se puede observar los paréntesis se encuentran vacíos. Al no pasar ninguna variable se toman los valores por defecto en la población. Para poder ver cuáles son las características propias de la población se debe ejecutar el siguiente código:

```
Population.info(poblacion)
```

```
## tamaño: 100
## ploidía: 2
## frecuencias alelicas iniciales: {'A': (0.5, 0.5), 'B': (0.5, 0.5)}
## desequilibrio de ligamiento: 0
## frecuencia de recombinacion: 0.5
```

```
## tasa de mutaciones: (0, 0)
## generación actual: 0
## sistema de determinación del sexo: XY
## tipo de seleccion: 0
```

Si lo que se quiere es modificar alguno (o todos) sus atributos, se puede hacer pasándole un nuevo valor en el proceso de creación de la población.

```
#se crean las variables que contienen el tamaño y las frecuencias iniciales
tam = 1000
frec_alel = {'A': (0.5,0.5), 'B':(0.5,0.5)}

# inicia la población
poblacion = Population(size=tam,
                       freq=frec_alel)
```

Aunque pueda parecer complejo lo que ocurre es que se asigna un nuevo valor a la población de frecuencias alélicas. En este caso, tanto el gen A como el gen B tendrán una frecuencia de 0.5 para ambos alelos en la población.

En estos momentos la población se encuentra vacía, es decir, se le han definido sus características pero no hay individuos en ella. Es por ello que es necesario llamar a un *método* del objeto población.

```
poblacion.initIndividuals()
```

```
## se han generado un total de 1000 individuos de la poblacion
```

Este método inicia a los individuos, que tendrán las características especificadas anteriormente.

En este caso, iniciará `rpy$tam‘` individuos que tendrán unos valores de frecuencias alélicas próximos a los especificados.

Para ver información sobre algunos de estos individuos se puede ejecutar el siguiente código

```
poblacion.printIndividuals(show=5)
```

```
## ide  sex sex_chromosome  chromosome
## g0.ID-0  Female  XX  {'c1': 'AB', 'c2': 'AB'}
## g0.ID-1  Female  XX  {'c1': 'aB', 'c2': 'Ab'}
## g0.ID-2  Male    XY  {'c1': 'aB', 'c2': 'ab'}
## g0.ID-3  Male    XY  {'c1': 'Ab', 'c2': 'Ab'}
## g0.ID-4  Male    XY  {'c1': 'aB', 'c2': 'AB'}
```

Una vez iniciados los individuos en la población se considera que éstos son la generación número 0.

El siguiente paso es llevar a cabo la evolución de la población

Table 1: Cambio en las frecuencias alélicas a lo largo de las generaciones

	p(A)	p(B)
gen.0	0.51	0.51
gen.10	0.49	0.51
gen.20	0.50	0.51
gen.30	0.51	0.48
gen.40	0.50	0.50
gen.50	0.46	0.46
gen.60	0.44	0.46
gen.70	0.39	0.44
gen.80	0.34	0.49
gen.90	0.33	0.49
gen.100	0.26	0.47

```
poblacion.evolvePop(gens=100)
```

```
## 10.0% completado...
## 20.0% completado...
## 30.0% completado...
## 40.0% completado...
## 50.0% completado...
## 60.0% completado...
## 70.0% completado...
## 80.0% completado...
## 90.0% completado...
## ¡Evolucion completada!
```

El parámetro más importante de `evolvePop` es `gens`, se trata del número de generaciones que tomará la evolución.

Tras la evolución el proceso ya queda completado, sin embargo la información no queda accesible. Para ello es posible obtener una tabla de valores o una representación gráfica.

Si lo que se busca es una tabla de valores, más conocido como *dataframe*, se puede recurrir al siguiente código:

```
allelicF = poblacion.getDataFrame("frecuencias alelicas")
```

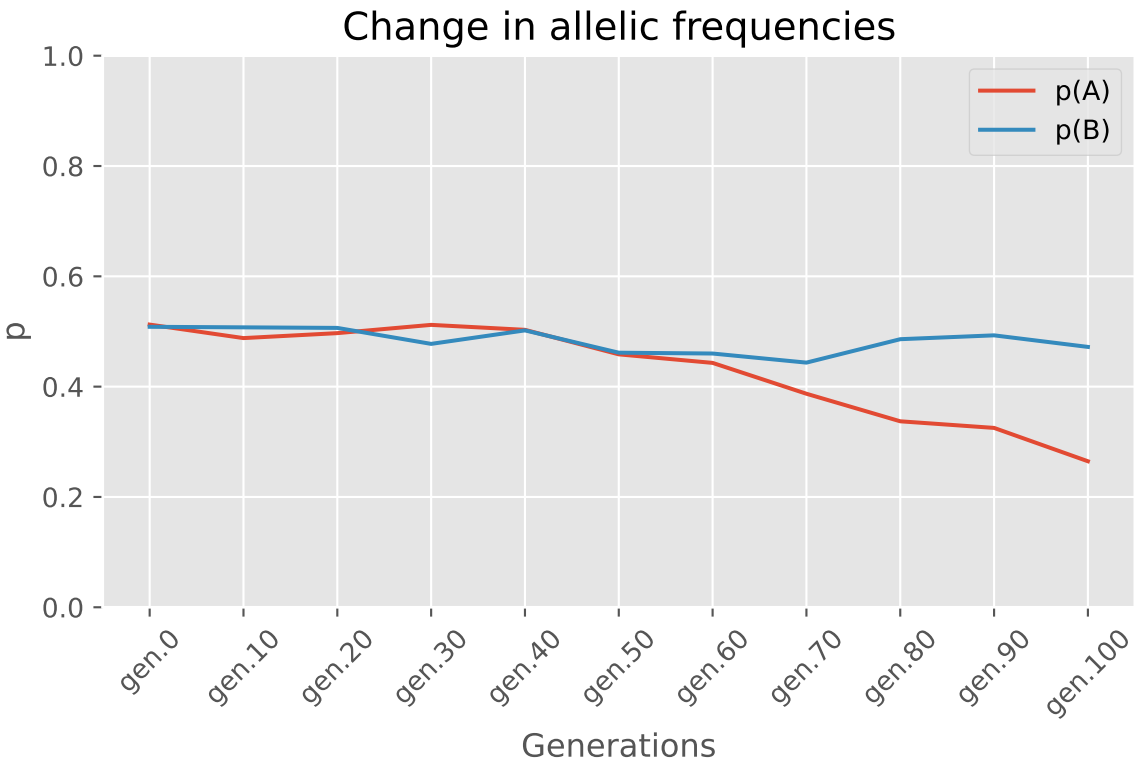
```
## Warning: package 'kableExtra' was built under R version 4.1.3
```

Como se indica dentro del paréntesis, el código devuelve una tabla donde quedan registrados todos los cambios que ha habido en las frecuencias alélicas tras el paso de las generaciones.

Si lo que se busca es obtener una representación gráfica de este proceso es necesario ejecutar el siguiente código:

```
from populy.plots import Plots
```

```
Plots.alleles(poblacion)
```



La sintaxis es diferente porque se utiliza otro módulo del paquete Populy (estructura).

Tal y como se observa, la población ha evolucionado ya que las frecuencias alélicas han sufrido cambios durante las generaciones.

Para este caso de uso en el que no se ha introducido al principio ningún parámetro de *fitness*,mutación ni recombinación y al no existir migración se puede decir que la fuerza evolutiva que actúa para producir el cambio ha sido la deriva genética. Tal y como se observa, aun con un tamaño de 1000 individuos la influencia de este mecanismo es apreciable y similar a lo que podría observarse en una población sometida a selección natural.

Este es el funcionamiento más básico, una vez comprendido es posible introducir una o más fuerzas de cambio evolutivo como son la mutación y la selección y estudiar su efecto, o el equilibrio entre ambas.

Para una explicación más detallada e interactiva es recomendable acudir a la Guía de uso LINK MYBINDEER, donde es posible ejecutar el código y trabajar con éste tal y como se ha pensado su uso.

Caso práctico: la deriva genética

En el siguiente código se llevará a cabo la simulación y visualización del efecto de la deriva genética en el cambio evolutivo.

La deriva genética es el proceso de cambio debido al error aleatorio del muestreo, al tratarse de una consecuencia de la reproducción aleatoria y muestreo su efecto se ve magnificado en poblaciones de tamaños reducidos, la consecuencia observable de la deriva es la fijación alélica y la pérdida de heterocigosidad promedio.

Para este ejemplo se utilizará un módulo de Populy llamado Superpop, este módulo permite simular de forma simultánea varias poblaciones con las mismas características, haciendo que los efectos de la deriva genética puedan ser observados con mayor claridad.

Primero, al igual que con `Population`, es necesario importar el módulo:

```
from populy.superpop import Superpop
```

La instanciación ocurre de la misma manera, se le pueden pasar los mismos parámetros que a `Population`pero es necesario indicarle el número `n` de poblaciones que se simularán.

```
super_small = Superpop(n=6,popsize=10)
```

En este caso se crean 6 poblaciones, cada una con un tamaño de 10 individuos.

Tras la instanciación, se debe ejecutar el siguiente código, que condensa las operaciones de iniciación y evolución de la población anteriormente vistas.

Aunque es posible obtener un *dataframe* de cambios en las frecuencias alélicas; es mucho más recomendable representar gráficamente el resultado.

```
super_small.plotPops()
```

Como se puede observar, el efecto de la deriva ocasiona un cambio en las frecuencias alélicas para ambos locus.

Cabe la pregunta de ¿qué ocurre en poblaciones de mayor tamaño? Para eso es posible llevar a cabo el proceso anterior pero cambiando el valor de `popsize`. Para compactar el código se ha realizado mediante los siguientes bloques:

En esta parte se crea una lista de 2 valores, que serán el tamaño de cada población. Posteriormente se crean y almacenan ambas poblaciones en una lista.

```
sizes = [100,1000]

pops = [Superpop(n=6,popsize=tam) for tam in sizes]
```

Tras esto se lleva a cabo, para cada población en `Pops` el proceso de iniciación y evolución.

Por último se representan en dos gráficos independientes los valores obtenidos de frecuencias alélicas para cada locus.

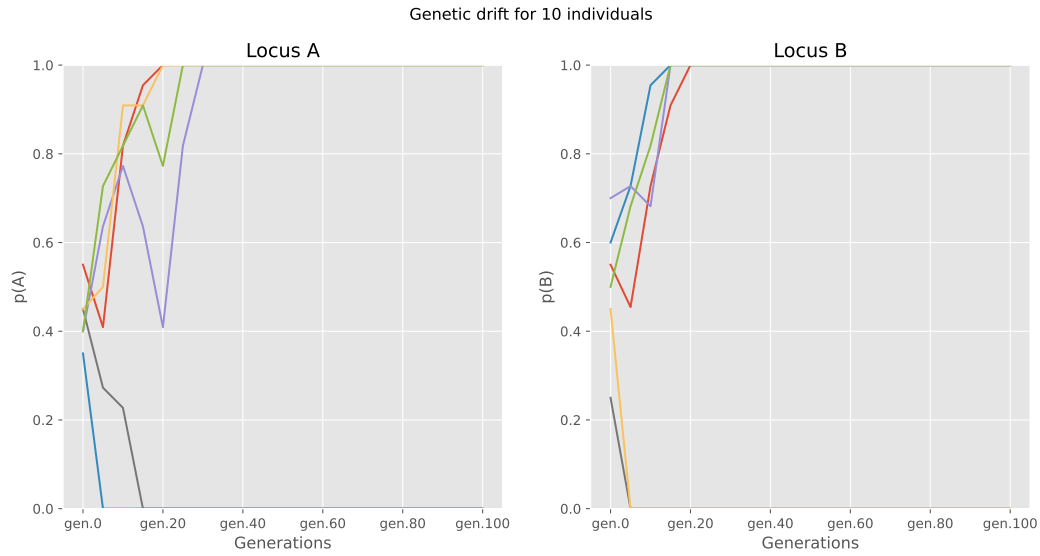
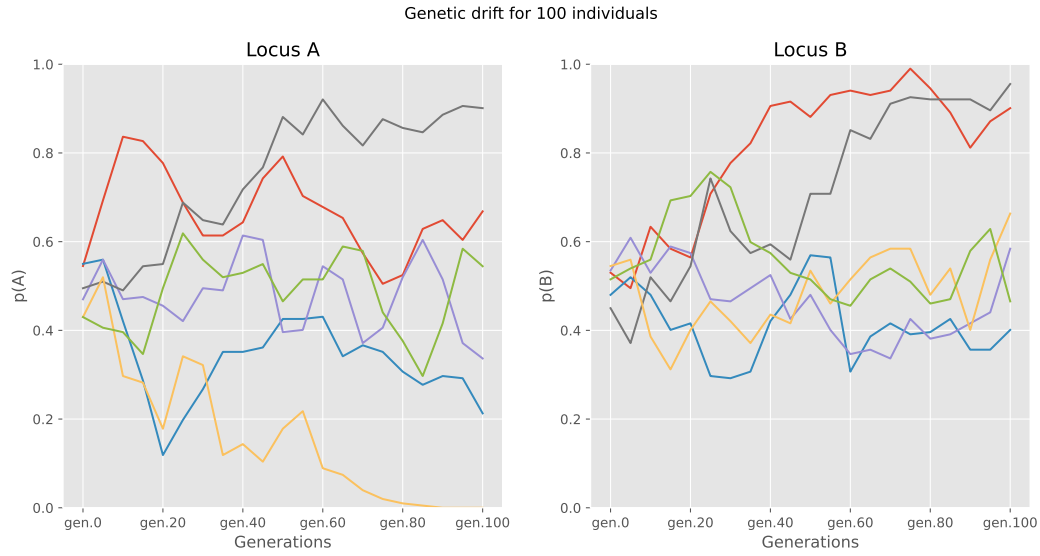
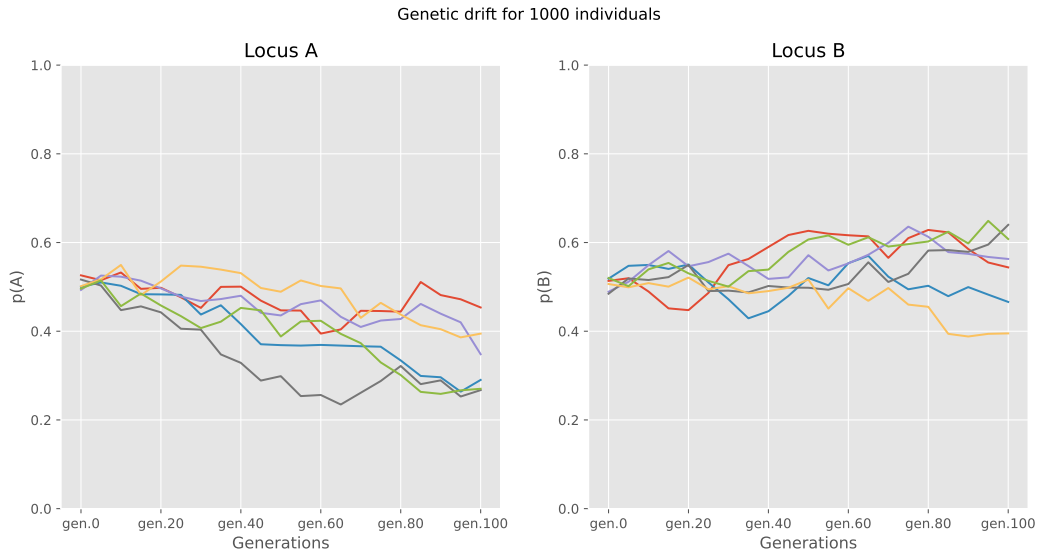


Figure 5: Representación gráfica de los efectos de la deriva genética para las frecuencias alélicas de dos locus A (izq.) y B (der.) de 6 poblaciones con 10 individuos

```
for x in pops:
  x.plotPops()
```





Se puede observar el efecto esperado, a mayor tamaño de población menor es el efecto de la deriva genética y menor es la desviación con respecto al valor esperado en ausencia de otra fuerza evolutiva.

Estructura

Populy comienza como una idea de aproximar los conceptos de evolución y la programación a alumnos de la rama de la biología debido a las deficiencias observadas en los programas utilizados como Populus, que se han descrito en apartados anteriores. El objetivo es que los alumnos sean partícipes activos en el estudio de los procesos evolutivos mientras aprenden algunos fundamentos básicos de la programación, enfocada en Python.

Al comienzo se definieron los objetivos generales del paquete, que son:

1. Obtener un software funcional.
2. Que un usuario sin experiencia en programación pueda comprender qué está sucediendo.
3. Obtener unos resultados, preferiblemente en formato gráfico, que permitan visualizar el fenómeno estudiado.

Más tarde, se comenzó con el desarrollo del software, en esta etapa es necesario establecer un esquema o sistema general. Se pueden establecer tres entidades esenciales a la hora de desarrollar este tipo de simulaciones: la Población, el individuo y el genoma/cromosoma. Desde un principio quedó claro que el usuario debía poder interactuar de forma sencilla con la población, ya que es el objeto de estudio, los objetos individuos y genoma serian características las cuales el usuario no debía interactuar directamente.

Con la ventaja de ser un lenguaje multiparadigma, Python permite definir una estructura de programación muy flexible y poco restrictiva. Se eligió el paradigma orientado a objetos (OOP) debido a la idoneidad de modelar una población creando una plantilla para la creación de objetos con unas características y funciones determinadas. A estas plantillas, en el paradigma de orientación a objetos se les conoce como Clases, cada clase representa una entidad, objeto o concepto determinado. Cada clase está compuesta por unas características y unos comportamientos que la definen. A las

características se les conoce como atributos y a los comportamientos como métodos (o funciones de clase/instancia).

En el caso de Populy debían existir, como mínimo, dos clases diferentes: la población y el individuo. A partir de este modelo de clases un usuario debía poder crear un nuevo objeto que pertenezca a una de estas clases, esto es, una población determinada. El proceso de crear un nuevo objeto se le conoce como instanciación. Tras crear el objeto y pasarle los valores que debe tener esa población el usuario debería llamar a los métodos correspondientes para llevar a cabo el proceso de evolución, entre otros.

La forma de empaquetar estas estructuras de clase se le conoce como crear un paquete, un paquete está compuesto principalmente de dos tipos de archivo: módulos e init. Un módulo contendrá la estructura de una clase en particular y tiene extensión `.py`, el archivo init, indica al intérprete de Python que ese directorio se debe tratar como un paquete y no como módulos separados.

Así pues, el desarrollo comienza creando 2 módulos dentro del paquete Populy, llamados *population* e *individual*, estos módulos contienen las clases Population e Individual, respectivamente.

Population debía tener los atributos generales de la población y métodos que permitieran llevar a cabo los diferentes procedimientos para iniciar, evolucionar y obtener información de ella.

Individual debía tener atributos particulares de cada individuo y también generales, de la población a la cual pertenecen, así como métodos que permitan crear ese individuo desde 0 o a partir de unos padres.

Más tarde se incluyó un nuevo módulo con una clase llamada Superpop, donde se recogería la capacidad de realizar simulaciones de varias poblaciones al mismo tiempo.

El resultado queda reflejado en el siguiente diagrama de clases, donde se puede observar los atributos y métodos más relevantes de cada clase:

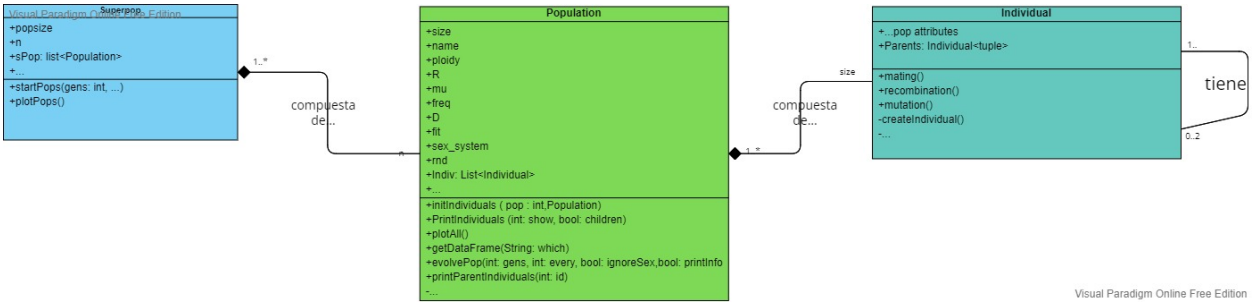


Figure 6: diagrama de clases de Populy

Tal y como se observa en el diagrama, las clases se relacionan entre ellas mediante una composición, esto es que la clase Superpop contiene objetos de la clase Population, que a su vez contiene objetos de la clase individual. Esto es similar a una población normal, que está compuesta de individuos. Debe distinguirse dos peculiaridades, la primera es que la clase Superpop no es un conjunto de poblaciones que interaccionan, se trata de una clase que permite agrupar la creación de multitud de poblaciones similares pero no interaccionan entre ellas. La segunda se trata de una relación entre los propios individuos, esto se debe a que un individuo tiene unos progenitores, que son a su vez individuos.

Además de estos módulos, el paquete también contiene unos tests unitarios, que sirven para hacer comprobaciones automáticas de varios casos posibles y casos límites a la hora de utilizar el paquete. Esto es útil para encontrar errores o comprobar cambios que se han realizado en el paquete durante su creación.

Explicar la diferencia entre paquete y script, describir todos los archivos y la función que cumplen. Centrarse en el directorio populy, explicar detalladame que hace cada modulo, cuales son sus atributos y sus metodos, o las funciones que contiene. Al menos las mas relevantes para comprender el paquete. Hablar de los tests y su importancia para un buen desarrollo de software.

Perspectiva y limitaciones

Populy es un paquete de simulación de procesos evolutivos desarrollado en Python con propósitos educativos. Uno de los objetivos de éste era crear un software acceso libre para que cualquier persona pueda utilizarlo. No solo esto, la idea es que el mismo paquete pueda sufrir modificaciones para incluir mejoras en el rendimiento, estructurales y nuevas simulaciones más avanzadas.

En estos momentos Populy tiene limitaciones en el rendimiento al simular grandes poblaciones con varios loci, sería interesante poder revisar la implementación en el código fuente en este aspecto para mejorar su tiempo de simulación.

Actualmente el genoma se encuentra codificado como un diccionario de pares clave-valor, donde la clave es el locus y el valor el alelo para ese locus. Esta forma, aunque sencilla y eficaz para pocos loci se convierte en una opción limitada al aumentar el número de genes. Existen alternativas como almacenar los cromosomas en un array de bits, mediante este método más compacto sería posible simular loci enteros según su secuencia genética, haciendo el modelo mucho más realista y permitiendo simulaciones más avanzadas, también haría posible simular más de dos alelos, así como mutaciones de un solo nucleótido e incluso delecciones e inserciones.

Otro obstáculo es el número de autosomas, en la actualidad Populy solamente soporta un único cromosoma. Aunque para simulaciones de pocos genes puede ser adecuado también le resta realismo a la simulación.

En cuanto a modelos, actualmente Populy cuenta con los necesarios para el aprendizaje de los procesos evolutivos más relevante: mutación, recombinación, selección, deriva genética, entre otros. Sin embargo sería necesario incluir la posibilidad de introducir nuevos genes en la población mediante la migración.

Conclusiones

Es una realidad que las herramientas tecnológicas están cambiando la forma que los alumnos aprenden, no solo en forma sino también en contenidos. La importancia de los ordenadores en las aulas no ha hecho más que crecer y con ello deben hacerlo las nuevas herramientas educativas.

Es esencial proporcionar al alumnado aquellas herramientas que permitan comprender mejor los contenidos de la asignatura y que se adapten a los cambios tecnológicos. En ocasiones estos cambios son complejos pero no por ello se debe dejar de explorar las posibilidades que ofrecen los nuevos medios tecnológicos de enseñanza.

Populy ha sido construido mediante la programación, a partir de los conceptos de la evolución y la genética de poblaciones pero siempre orientada a la enseñanza dinámica e interactiva. Como tal, Populy y sus *jupyter notebooks* reivindican que los métodos más anticuados de enseñanza como son el uso de programas del estilo de Populus ya no tienen sentido en las aulas; Populy pretende ser una herramienta actualizada y adaptada a las nuevas necesidades y avances técnicos para modelar procesos simples.

Para concluir, el trabajo está orientado para estudiantes en los primeros cursos de las ciencias biológicas y ciencias de la vida. En primer lugar porque la evolución es una pieza fundamental en estas enseñanzas y como tal requiere ser abordada en las aulas desde múltiples perspectivas ya sea teóricas, experimentales o computacionales; en segundo lugar porque pretende introducir conceptos sencillos de programación, una habilidad muy demandada y útil, en especial con los avances que tienen lugar en la biocomputación.

Referencias

Adamack, Aaron T., and Bernd Gruber. 2014. "PopGenReport: Simplifying Basic Population Genetic Analyses in r." *Methods in Ecology and Evolution* 5: 384–87. https://doi.org/10.1111/2041-210X.12158.

"Anaconda Software Distribution." 2020. *Anaconda Documentation*. Anaconda Inc. https://docs.anaconda.com/.

Bergmann, Ralph, and Wolfgang Wilke. 1996. "On the Role of Abstraction in Case-Based Reasoning." *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1168: 28–43. https://doi.org/10.1007/BFB0020600.

Carroll, Robert. 2000. "Carroll, r. L. Towards a New Evolutionary Synthesis. Trends Ecol. Evol. 15, 27-32." *Trends in Ecology & Evolution* 15 (February): 27–32. https://doi.org/10.1016/S0169-5347(99)01743-7.

Carvajal-Rodriguez, Antonio. 2008. "Simulation of Genomes: A Review." *Current Genomics* 9 (May): 155–59. https://doi.org/10.2174/138920208784340759.

Chen, Ouhao, Slava Kalyuga, and John Sweller. 2015. "The Worked Example Effect, the Generation Effect, and Element Interactivity." *Journal of Educational Psychology* 107 (August): 689–704. https://doi.org/10.1037/edu0000018.

Darwin, Charles. 1859. "The Origin of Species by Means of Natural Selection."

Dawkins, Richard. 1989. *The Selfish Gene*. New edition. Oxford ; New York : Oxford University Press, 1989. https://search.library.wisc.edu/catalog/999612476002121.

Drexel, Marisa. 2009. "Educational Psychology."

Iskrenovic-Momcilovic, Olivera. 2018. "Using Computers in Teaching in Higher Education." *Print) Mediterranean Journal of Social Sciences* 9: 2039–2117. https://doi.org/10.2478/mjss-2018-0116.

Jayal, Ambikesh, Stasha Lauria, Allan Tucker, and Stephen Swift. 2011. "Python for Teaching Introductory Programming: A Quantitative Evaluation." *Innovation in Teaching and Learning in Information and Computer Sciences* 10 (1): 86–90. https://doi.org/10.11120/ital.2011.10010086.

Johnson, Fionnuala, and Stephen McQuistin. 2020. "Analysis of Student Misconceptions Using Python as an Introductory Programming Language." *Proceedings of the 4th Conference on Computing Education Practice 2020*. https://doi.org/10.1145/3372356.

Jupyter, Project, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, et al. 2018. "Binder 2.0 - Reproducible, Interactive, Sharable Environments for

Science at Scale." In, 113–20. https://doi.org/10.25080/Majora-4af1f417-011.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows." Edited by F. Loizides and B. Schmidt. IOS Press.

Koetsier, Teun. 2001. "On the Prehistory of Programmable Machines: Musical Automata, Looms, Calculators." *Mechanism and Machine Theory* 36 (May): 589–603. https://doi.org/10.1016/S0094-114X(01)00005-2.

Liao, Yuen-Kuang Cliff, and George W. Bright. 1995. "Effects of Computer Programming on Cognitive Outcomes: A Meta-Analysis:" *Http://Dx.doi.org/10.2190/E53g-Hh8k-AJRR-K69m* 7 (January): 251–68. https://doi.org/10.2190/E53G-HH8K-AJRR-K69M.

Masters, J. C., and H. G. Spencer. 1989. "Why We Need a New Genetic Species Concept." *Systematic Biology* 38 (September): 270–79. https://doi.org/10.2307/2992287.

Messer, Philipp W. 2013. "SLiM: Simulating Evolution with Selection and Linkage." *Genetics* 194 (4): 1037–39.

Mishra, Punya, Aman Yadav, Danah Henriksen, Kristen Kereluik, Laura Terry, Chris Fahnoe, and Colin Terry. 2013. "Rethinking Technology & Creativity in the 21st Century." *TechTrends* 57 (May): 10–14. https://doi.org/10.1007/S11528-013-0655-Z.

Muirhead, C. A. 2016. "Genetic Drift, Models of Random." *Encyclopedia of Evolutionary Biology*, January, 136–43. https://doi.org/10.1016/B978-0-12-800049-6.00024-X.

Nola, Robert, and Gurol Irzik. 2005. *Philosophy, Science, Education and Culture*. 2005th ed. Contemporary Trends and Issues in Science Education. New York, NY: Springer.

Nordborg, Magnus. 2000. "Coalescent Theory."

"Object-Oriented Programming." 2022. *Wikipedia*. Wikimedia Foundation. https://en.wikipedia.org/wiki/Object-oriented_programming.

Okasha, Samir. 2016. "Population Genetics." In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Winter 2016. https://plato.stanford.edu/archives/win2016/entries/population-genetics/; Metaphysics Research Lab, Stanford University.

Peng, Bo, and Marek Kimmel. 2005. "simuPOP: A Forward-Time Population Genetics Simulation Environment." *Bioinformatics (Oxford, England)* 21 (September): 3686–87. https://doi.org/10.1093/BIOINFORMATICS/BTI584.

Peng, Bo, Marek Kimmel, and Christopher I Amos. 2010. *Forward-Time Population Genetics Simulations*. Hoboken, NJ: Wiley-Blackwell.

R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Servedio, Maria R., Yaniv Brandvain, Sumit Dhole, Courtney L. Fitzpatrick, Emma E. Goldberg, Caitlin A. Stern, Jeremy Van Cleve, and D. Justin Yeh. 2014. "Not Just a Theory—the Utility of Mathematical Models in Evolutionary Biology." *PLoS Biology* 12: 1002017. https://doi.org/10.1371/JOURNAL.PBIO.1002017.

Shute, Valerie J., Chen Sun, and Jodi Asbell-Clarke. 2017. "Demystifying Computational Thinking." *Educational Research Review* 22 (November): 142–58. https://doi.org/10.1016/J.EDUREV.2017.09.003.

Soler, Manuel. 2002. *Evolución : La Base de La Biología*. Proyecto Sur de Ediciones.

"Stack Overflow Developer Survey 2021 -Popular Technologies." n.d. *Stack Overflow*. https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies.

Sweller, John, and Graham A. Cooper. 1985. "The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra." *Cognition and Instruction* 2 (1): 59–89. https://doi.org/10.1207/s1532690xci0201/_3.

Teaching and Learning with Jupyter. 2022. https://jupyter4edu.github.io/jupyter-edu-book/.

Van Rossum, Guido, and Fred L. Drake. 2009. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Wing, Jeannette M. 2008. "Computational Thinking and Thinking about Computing." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366 (October): 3717–25. https://doi.org/10.1098/RSTA.2008.0118.

Appéndice