Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

PyToxo: una herramienta Python para calcular tablas de penetrancia de modelos de epistasia de alto orden

Estudiante: Borja González Seoane
Dirección: María José Martín Santamaría
Dirección: Christian Ponte Fernández

A Coruña, junio de 2021

A mis padres Chito y María, por todo

Agradecimientos

En primer lugar me parece necesario reconocer a los directores de este trabajo, María José Martín Santamaría y Christian Ponte Fernández, ya que su guía y su atenta disposición fueron completamente determinantes en el transcurso del mismo. También quiero recordarme de Jorge González Domínguez.

Al resto de profesores de la Facultade de Informática de A Coruña, por los valiosos conocimientos que me han impartido a lo largo de estos años y que en parte espero poder plasmar en este proyecto.

Sobre todo, no puedo dejar de agradecerle a mi familia el apoyo incondicional que me ha brindado absolutamente siempre. Este manuscrito pone fin a una etapa de mi vida que ni tan siquiera hubiera podido iniciarse de no ser por ellos.

Quiero trasladarle mi gratitud a mis amigos, a los de toda la vida y a los que he conocido en este periplo en la facultad. En especial, mencionar a los compañeros del grupo 2.3 del primer año de carrera, que siempre se han volcado en ayudarme en todo en lo que han podido.

Por último, a toda la gente que directa o indirectamente me ha ayudado en el transcurso de mis estudios. Este trabajo va, humildemente, por todos ellos.

Resumen

Los estudios de asociación del genoma completo analizan marcadores genéticos con el fin de poder asociar una determinada variación genética a una enfermedad. Los estudios de asociación contemporáneos se centran en interacciones epistáticas, pues es sabido que el análisis individual de los locus no resulta suficiente para explicar muchas de las enfermedades del mundo real. La epistasia es el fenómeno por el que la expresión de un determinado gen se ve afectada por la de otros genes independientes, es decir, hace referencia a situaciones más complejas en las que varios genes diferentes se influyen recíprocamente.

La forma más común de representar interacciones epistáticas es a través de lo que se conoce como tablas de penetrancia. No obstante, aunque la mayoría de simuladores que permiten crear conjuntos de datos de casos-controles para estudios de asociación admiten el uso de estas tablas, en general no pueden crearlas o tienen bastantes limitaciones a la hora de hacerlo.

Este Trabajo de Fin de Grado ha conllevado la creación de PyToxo, una herramienta Python para calcular tablas de penetrancia de modelos de epistasia de alto orden. PyToxo es una reimplementación de Toxo, una librería anterior que introdujo una aproximación matemática innovadora en el estado del arte, pero que también presentaba una serie de carencias y puntos débiles que fueron subsanados con esta nueva herramienta.

Con la conclusión de este Trabajo de Fin de Grado se ha alcanzado un *software* que mejora a Toxo —y con él al resto de herramientas del estado del arte— en materia de alcance, precisión, tiempo de ejecución y usabilidad. Estos resultados tan satisfactorios permiten presentar a la comunidad científica una utilidad interesante para el ámbito de investigación de los estudios de asociación del genoma completo, con la que pueden conseguirse datos sintéticos más realistas con los que continuar indagando en la dimensión genética de las enfermedades.

Abstract

Genome-wide association studies analyze genetic markers in order to be able to associate a certain genetic variation to a disease. Contemporary genome-wide association studies focus on epistatic interactions, as it is known that the analysis of individual locus is not sufficient to explain many real-world diseases. Epistasis is the phenomenon by which the expression of a certain gene is affected by that of other independent genes, that is, it refers to more complex situations in which several different genes interact.

The most common way to represent epistatic interactions is through what are known as penetrance tables. However, although most simulators that allow creating case-control data

sets for association studies allow the use of these tables, in general they cannot create them or can do so with many limitations.

This Final Degree Dissertation has led to the creation of PyToxo, a Python tool to calculate penetrance tables for high-order epistasis models. PyToxo is a reimplementation of Toxo, an earlier library that presented an innovative mathematical approach to the state of the art, but that also brings its own shortcomings, corrected in this new application.

With the conclusion of this Final Degree Dissertation we have obtained a software that improves Toxo —and with it the rest of the state of the art tools— in terms of scope, accuracy, runtime and usability. These very satisfactory results make it possible to present to the scientific community an interesting utility in the research field of genome-wide association studies, with which more realistic synthetic data can be obtained to continue investigating the genetic dimension of diseases.

Palabras clave:

- Tablas de penetrancia
- Epistasia
- Genoma completo
- Genoma humano
- Python
- Genética
- Bioinformática

Keywords:

- Penetrance tables
- Epistasis
- Full genome
- Human genome
- Python
- Genetics
- Bioinformatics

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Índice de pseudocódigos

Capítulo 1

Introducción

En este primer capítulo se describe la motivación y los objetivos principales que se afrontan con este Trabajo de Fin de Grado. También se presenta la estructura que seguirá la presente memoria.

1.1 Motivación

Este Trabajo de Fin de Grado está enmarcado en el campo de los estudios de asociación del genoma completo o en inglés *genome-wide association studies* (GWAS). Estos estudios buscan diferencias en marcadores genéticos entre grupos caso-control que puedan explicar determinados fenotipos con el fin último de poder asociar una determinada variación genética a una enfermedad.

Los marcadores genéticos que se suelen utilizar en los GWAS son los polimorfismos de un solo nucleótido o en inglés *single nuclotide polymorphisms* (SNP). Un SNP identifica una posición específica (locus) en el genoma donde al menos un 1 % de la población presenta una variación genómica. La mayoría de los SNP son bialélicos, es decir, pueden tomar dos valores diferentes, A o a, por lo que para cada SNP hay tres posibles genotipos: AA, Aa y aa.

Inicialmente sólo se consideraba la diferencia de frecuencia individual de cada SNP entre los grupos de caso y control. Sin embargo, se ha observado que el análisis individual de los SNP no es suficiente para fenotipos complejos [43], por lo que se hace necesario estudiar la interacción entre ellos, lo que se conoce como epistasia.

La epistasia es un fenómeno genético que se define por una interacción de la variación genética en dos o más *loci* para producir un resultado fenotípico que no se predice por la combinación aditiva de efectos atribuibles a los *loci* individuales [4]. En el marco particular de este trabajo, la epistasia hace referencia a la interacción entre dos o más SNP para un fenotipo específico, donde el número de *loci* implicados define el orden de la interacción.

Existen en la bibliografía diferentes simuladores que permiten crear conjuntos de datos

de casos-controles para estudios de GWAS. Algunos ejemplos son GAMETES [50], EpiSIM [38] o GenomeSIMLA [5]. Estos simuladores permiten definir una relación epistática entre los datos. La forma más habitual de describir esa relación es a través de lo que se conoce como tabla de penetrancia. Una tabla de penetrancia captura la probabilidad de expresar el fenotipo a estudiar dada una combinación de alelos en particular.

No obstante, a día de hoy la mayoría de simuladores *software* disponibles sólo pueden crear tablas de penetrancia para modelos de epistasia conocidos que involucran un pequeño número de genes y necesitan satisfacer abundantes limitaciones.

En Ponte-Fernández *et al.* [21] se introduce Toxo, una librería de MATLAB [46] diseñada para calcular tablas de penetrancia de modelos de epistasia. Toxo es capaz de calcular tablas de penetrancia a partir de modelos que contienen funciones de penetrancia bivariadas sin limitaciones en el orden de interacción, permitiendo además utilizar valores más realistas de parámetros como la heredabilidad. La heredabilidad es la parte de variación fenotípica de una población que depende del genotipo de cada individuo. La mayoría de los simuladores presentes en la bibliografía utilizan valores de heredabilidad por debajo de 0,20 para tablas de penetrancia de alto orden. Sin embargo, las enfermedades del mundo real presentan valores de heredabilidad mayores [48, 52].

Aunque Toxo resuelve varias de las carencias de los simuladores del estado del arte, también presenta sus propias limitaciones. Es aquí donde surge la motivación de este Trabajo de Fin de Grado, y con él de PyToxo, un *software* con el que pretendemos plantear una evolución de Toxo basada en su aproximación matemática y corrigiendo algunas de sus problemáticas identificadas.

PyToxo será un nuevo producto *software* escrito en Python, que constituirá una librería y a la vez una herramienta final orientada al cálculo de tablas de penetrancia para modelos de epistasia de alto orden.

1.2 Objetivos

El principal objetivo de este Trabajo de Fin de Grado es el rediseño e implementación en el lenguaje de programación Python [30] de un producto *software* que emule la funcionalidad de Toxo, mejorando además algunas de sus prestaciones. Como ya hemos avanzado, este producto *software* recibirá el nombre de PyToxo.

El primer problema que presenta Toxo para un usuario final es que, al estar implementado en MATLAB [46], requiere que el usuario tenga MATLAB instalado para trabajar con él. MATLAB es un *software* comercial de pago y el usuario necesitará una licencia del mismo para poder ejecutar Toxo. Sin embargo, una implementación en Python como la que planteamos con PyToxo puede distribuirse como *software* libre, al no haber restricciones al respecto ni

para con la herramienta en particular ni para con el lenguaje en sí, que también es libre.

Por otra parte, con PyToxo pretendemos mejorar la usabilidad del programa final, presentando interfaces más amigables. Este propósito abarca una interfaz programática en forma de librería —para poder ser utilizada en otros programas Python fácilmente—, una interfaz en línea de comando —para usuarios avanzados o para poder ejecutar el programa en modo de procesamiento por lotes (*batch*)— y una interfaz gráfica —especialmente orientada para usuarios poco familiarizados con entornos de ejecución en línea de comandos—.

Además de lo anterior, con PyToxo también pretendemos mejorar la precisión numérica del cálculo de las tablas de penetrancia que manifiesta Toxo y ampliar la complejidad soportada para calcular dichas tablas.

Finalmente, una vez concluido el trabajo descrito, se liberará y publicitará PyToxo para que pueda ser utilizado por la comunidad científica. Para ello, PyToxo será distribuido a través de un repositorio público, permitiendo una fácil instalación a sus usuarios finales.

1.3 Estructura de esta memoria

La presente memoria se inicia con un capítulo introductorio, que pone de manifiesto la motivación de este Trabajo de Fin de Grado y los objetivos a perseguir. Seguidamente, afrontaremos la base matemática del cálculo de las tablas de penetrancia y el estado del arte en cuanto a simuladores que las explotan. En el capítulo 3, nos centraremos en la descripción detallada de PyToxo a nivel *software*. A continuación, dedicaremos un capítulo a presentar las diferentes interfaces con las que se ha provisto a la herramienta. En el quinto capítulo defenderemos los resultados obtenidos con este proyecto. Por último, expondremos las conclusiones alcanzadas y algunas líneas de trabajo futuro.

Capítulo 2

Cálculo de las tablas de penetrancia

Este capítulo se centra en abordar las diferentes aproximaciones matemáticas posibles para el cálculo de las tablas de penetrancia, a la vez que se presentan algunos simuladores del estado del arte que emplean estas tablas. Seguidamente se describe en detalle, a nivel matemático, Toxo, la librería MATLAB capaz de calcular tablas de penetrancia para modelos epistáticos de cualquier orden de interacción.

2.1 Tablas de penetrancia

Como ya avanzábamos en el Capítulo 1, una tabla de penetrancia relaciona las diferentes combinaciones de alelos que pueden propiciar la expresión de un determinado fenotipo, junto con la probabilidad asociada de manifestar este último. En la Tabla 2.1 mostramos un ejemplo de una.

Las tablas de penetrancia constituyen la forma más común de caracterizar relaciones epistáticas, y a día de hoy la mayoría de los simuladores de datos genéticos de la bibliografía las utilizan. No obstante lo anterior, no todos ellos permiten generar las tablas. Simuladores como SimuPOP [18], SBVB [35] o HapSample [51] no pueden crear tablas de penetrancia, sino solamente simular conjuntos de datos a partir de las mismas.

Otros simuladores del estado del arte, como detallaremos a continuación en la siguiente sección, sí permiten calcular las tablas, pero con considerables limitaciones.

	BB	Bb	bb
AA	0.00031	0.00031	0.00031
Aa	0.00031	0.00231	0.01750
aa	0.00031	0.01750	1

Tabla 2.1: Ejemplo de una tabla de penetrancia para dos marcadores bialélicos

	BB	Bb	bb
AA	α	$\alpha(1+\theta)$	$\alpha(1+\theta)^2$
Aa	$\alpha(1+\theta)$	$\alpha(1+\theta)^2$	$\alpha(1+\theta)^3$
aa	$\alpha(1+\theta)^2$	$\alpha(1+\theta)^3$	$\alpha(1+\theta)^4$

Tabla 2.2: Ejemplo de modelo de Marchini *et al.*: modelo aditivo de orden 2

2.2 Cálculo de las tablas de penetrancia

Existen tres enfoques generales para crear tablas de penetrancia. El primero y más sencillo consiste en utilizar un modelo epistático, que es una relación matemática que define, en función de una o más variables, el valor de penetrancia para cada combinación de genotipos. Estas variables suelen representar un parámetro estadístico de la interacción.

En Marchini *et al.* [13] se proponen unos modelos muy conocidos y ampliamente utilizados en GWAS. Estos modelos emplean dos parámetros para definir la interacción entre *loci*: el efecto base (α) y el efecto genotípico (θ). El efecto base es el efecto genético presente en cada locus independientemente de la combinación de alelos real; mientras que el efecto genotípico es el aumento de las probabilidades de una determinada enfermedad más allá del efecto base, y debido a la interacción genética. La Tabla 2.2 muestra un ejemplo de un modelo de orden 2.

A partir de los modelos epistáticos se puede obtener una tabla de penetrancia dando valores a cada uno de los dos parámetros. Sin embargo, estos parámetros deberán satisfacer ciertas restricciones para poder ser utilizados, pues las penetrancias son probabilidades, y por ello sólo pueden tomar valores dentro del intervalo $[0, 1]$.

El segundo método consiste en definir un conjunto de características que debe cumplir la población simulada en el estudio y encontrar una tabla de penetrancia que encaje con ellas. Las características de la población más comunes a parametrizar son la prevalencia ($P(D)$), que representa la proporción de individuos en una población que posee el fenotipo a estudiar, y la heredabilidad (h^2), que representa la cantidad de variación fenotípica que corresponde a la variación genética. Hallar una tabla con tales requisitos es un proceso más complejo que usar un modelo epistático.

GAMETES [50] es un software de simulación epistática que utiliza un método estocástico para encontrar una tabla de penetrancia con los niveles de prevalencia y heredabilidad deseados. También puede generar muestras de población a partir de estas tablas. GenomeSIMLA [5] es otro simulador capaz de encontrar una tabla de penetrancia para unas restricciones de prevalencia y heredabilidad concretas, utilizando un algoritmo genético para llegar a una solución.

Por último, el tercer enfoque consiste en la combinación de los dos métodos anteriores: el uso de modelos de epistasia a la vez que de un conjunto de restricciones paramétricas sobre

características de la población. Este enfoque tiene la ventaja de modelar la interacción usando las variables del modelo y considerar también algunas características de la población por medio de las restricciones paramétricas. En contraprestación, calcular una tabla de penetrancia de esta forma es una tarea significativamente más compleja.

EpiSIM [38] es un simulador que parte de esta hibridación de métodos, permitiendo crear tablas de penetrancia y simular muestras de población a partir de ellas. Permite especificar valores de penetrancia en función de dos variables y también permite definir los valores deseados de prevalencia y heredabilidad. EpiSIM intenta hallar un valor para las variables del modelo resolviendo el sistema de ecuaciones formado por las expresiones de prevalencia y heredabilidad, definidas respectivamente a continuación:

$$P(D) = \sum_i P(D|g_i)P(g_i) \tag{2.1}$$

$$h^2 = \frac{\sum_i P(D|g_i) - P(D)^2 \sum P(g_i)}{P(D)\ 1 - P(D) \sum} \tag{2.2}$$

Donde:

- $P(D|g_i) = f_i(x, y)$ es la proporción de individuos que muestran el rasgo D cuando tienen el genotipo g_i.
- $P(g_i)$ es la frecuencia poblacional del genotipo g_i.
- $f_i(x, y)$ es la función de dos variables que define el modelo epistático.

Así, EpiSIM busca encontrar la tabla de penetrancia que cumpla las restricciones de prevalencia y heredabilidad resolviendo un sistema de ecuaciones compuesto por esas dos expresiones, resultando en un sistema con dos ecuaciones y dos incógnitas: las dos variables del modelo de epistasia (x e y).

Aunque teóricamente EpiSIM permite crear tablas de penetrancia de hasta cuarto orden, en la práctica funciona correctamente para modelos de segundo orden y valores bajos de heredabilidad y prevalencia, pero no puede encontrar soluciones para modelos de orden superior o valores de parámetros más altos. Además, dado que no todas las combinaciones de heredabilidad y prevalencia son posibles, este método tiende a construir como resultado sistemas de ecuaciones incompatibles.

En Ponte-Fernández *et al.* [21] se presenta Toxo, una librería de MATLAB capaz de calcular tablas de penetrancia a partir de modelos de cualquier orden que contienen funciones de penetrancia bivariadas y con un determinado valor de heredabilidad o prevalencia. Para superar las anteriores limitaciones, en lugar de encontrar una combinación específica de heredabilidad y prevalencia, Toxo permite al usuario definir o bien la heredabilidad o bien la

prevalencia, y maximiza el otro parámetro. En la siguiente sección detallamos minuciosamente la aproximación matemática de esta librería.

2.3 Método matemático de Toxo

La principal diferencia que plantea Toxo [21] frente a la aproximación de EpiSIM [38] es el maximizar uno de los dos parámetros disponibles, prevalencia o heredabilidad, permitiendo que el usuario fije el otro a voluntad.

Siguiendo este enfoque, la probabilidad de formular un sistema incompatible se reduce significativamente, ya que la mayoría de los modelos alcanzan todas las prevalencias y heredabilidades para alguna configuración concreta. Además, Toxo también considera el rango válido de valores de penetrancia como restricciones para el sistema de ecuaciones a resolver.

Toxo requiere que el modelo de entrada cumpla dos requisitos: ha de estar formado por expresiones monótonamente no decrecientes del espacio de los números reales positivos ($\mathbb{R}+$) y todas esas expresiones han de ser ordenables unívocamente en el mismo espacio $\mathbb{R}+$. Estas restricciones incluyen a la gran mayoría de los modelos utilizados en la literatura [21], como por ejemplo los ya mencionados modelos de Marchini ***et al.*** [13].

En caso de que el usuario decida fijar un valor concreto de heredabilidad, Toxo maximizará la prevalencia. Basándonos en la ecuación 2.1, maximizar la prevalencia implica maximizar la suma:

$$\sum_i P(D|g_i)P(g_i) \tag{2.3}$$

En este punto, es necesario precisar que, además de la heredabilidad o la prevalencia, el otro parámetro a especificar para el cálculo de la tabla de penetrancia por medio de este enfoque es la frecuencia del alelo menos común o en inglés *minor allele frequency* (MAF) asociada a cada locus. Suponiendo un equilibrio de ligamiento entre los *loci*, y bajo la ley de Hardy-Weinberg, la probabilidad de un genotipo se puede calcular como el producto de las probabilidades de cada alelo [15]. Así pues, fijando la MAF de cada locus, hacemos que $P(g_i)$ sea constante. Este principio se puede aplicar a cualquier orden de interacción.

Considerando lo anterior, y que las expresiones de penetrancia son monótonamente no decrecientes y ordenables, todas ellas se incrementarán proporcionalmente al incrementar sus variables. En consecuencia, su suma alcanzará su valor máximo cuando tome el suyo la expresión $P(D|g_i)$. Puesto que las penetrancias son probabilidades, su valor máximo es 1. Por lo tanto, podemos obtener la prevalencia máxima para un modelo, dado un valor de heredabilidad, resolviendo un sistema de ecuaciones formado por esta restricción de heredabilidad y la condición de prevalencia máxima:

$$\begin{aligned} \frac{\sum_i P(D|g_i) - P(D)\sum P(g_i)}{P(D)\ 1 - P(D)\sum} &= h^2 \\ max\ P(D|g_i)\sum &= 1 \end{aligned} \tag{2.4}$$

Se sigue un proceso análogo para maximizar la heredabilidad al fijar la prevalencia. En la Ecuación 2.2, el único término variable es la suma en el numerador, ya que la prevalencia y las MAF son fijas. Por lo tanto, para maximizarla necesitamos maximizar la suma:

$$\sum_i P(D|g_i) - P(D)\sum P(g_i) \tag{2.5}$$

Usando las mismas dos restricciones que antes, la suma será máxima cuando la expresión de penetrancia más grande tome su valor máximo. Así, nuevamente, podemos obtener la heredabilidad máxima para un modelo dado su valor de prevalencia resolviendo un sistema de ecuaciones compuesto por la expresión de la prevalencia y la condición de heredabilidad máxima:

$$\begin{aligned} \sum_i P(D|g_i)P(g_i) &= P(D) \\ max\ P(D|g_i)\sum &= 1 \end{aligned} \tag{2.6}$$

En ambos casos, un último paso necesario es descartar cualquier solución con valores negativos para cualquiera de las variables del modelo, pues las restricciones en los modelos sólo son verdaderas para números reales positivos.

Seguidamente proponemos, con fines explicativos, un ejemplo numérico completo de este proceso de cálculo.

2.3.1 Ejemplo numérico del método de Toxo

En la Tabla 2.2 mostramos un modelo de Marchini *et al.* [13], más concretamente el modelo aditivo de segundo orden de la susodicha colección. Partiremos de ese modelo para presentar en un ejemplo numérico el proceso completo de cálculo de una tabla de penetrancia válida, siguiendo el método matemático de Toxo [21].

En este ejemplo se maximizará, arbitrariamente, la heredabilidad, para unos valores también arbitrarios que fijaremos en 0,8 para la prevalencia y en $\frac{1}{3}$ para la MAF —para ambos genes implicados, pues es un modelo de segundo orden—.

Antes de proceder con el cálculo, hay que verificar que el modelo cumple los dos requisitos exigidos por Toxo que ya expusimos en la Sección 2.2: ha de estar formado por expresiones monótonamente no decrecientes del espacio de los números reales positivos ($\mathbb{R}+$) y todas esas expresiones han de ser ordenables unívocamente en el mismo espacio $\mathbb{R}+$.

	BB	Bb	bb
AA	16/81	16/81	4/81
Aa	16/81	16/81	4/81
aa	4/81	4/81	1/81

Tabla 2.3: Probabilidades de los genotipos para dos combinaciones de *loci*, ambos con la misma MAF $= \frac{1}{3}$

En primer lugar podemos comprobar que todos los miembros del modelo son expresiones monótonamente no decrecientes en $\mathbb{R}+$, pues sus derivadas parciales no presentan un cambio de signo para todo $x > 0$ e $y > 0$, como podemos observar a continuación:

$$\begin{aligned}
&\tfrac{\partial}{\partial x}\ x\textstyle\sum = 1\\
&\tfrac{\partial}{\partial y}\ x\textstyle\sum = 0\\
&\tfrac{\partial}{\partial x}\ x(1+y)\textstyle\sum = 1+y\\
&\tfrac{\partial}{\partial y}\ x(1+y)\textstyle\sum = x\\
&\tfrac{\partial}{\partial x}\ x(1+y)^2\textstyle\sum = (1+y)^2\\
&\tfrac{\partial}{\partial y}\ x(1+y)^2\textstyle\sum = x(2y+2)\\
&\tfrac{\partial}{\partial x}\ x(1+y)^3\textstyle\sum = (1+y)^3\\
&\tfrac{\partial}{\partial y}\ x(1+y)^3\textstyle\sum = x(3y^2+6y+3)\\
&\tfrac{\partial}{\partial x}\ x(1+y)^4\textstyle\sum = (1+y)^4\\
&\tfrac{\partial}{\partial y}\ x(1+y)^4\textstyle\sum = x(4y^3+12y^2+12y+4)
\end{aligned} \tag{2.7}$$

Por otra parte, en la siguiente relación se verifica que todas las expresiones también son ordenables unívocamente en $\mathbb{R}+$, con lo que se justifica también que ambos criterios son satisfechos por el modelo.

$$\begin{aligned}
&x \le x(1+y) \le x(1+y)^2 \le x(1+y)^3 \le x(1+y)^4,\\
&\forall x, y \in \mathbb{R}_{>0}
\end{aligned} \tag{2.8}$$

El siguiente paso es calcular la probabilidad asociada a cada combinación de dos genotipos. Como ya hemos visto, bajo la ley de Hardy-Weinberg, la probabilidad de un genotipo se puede calcular como el producto de las probabilidades de cada alelo. Así pues, dada la MAF $= \frac{1}{3}$ para los dos *loci*, las probabilidades de cada alelo son $p = \frac{1}{3}$ y $q = 1 - p = \frac{2}{3}$. Si la frecuencia del alelo a es p y la frecuencia del alelo A es q, entonces la probabilidad de tener el genotipo AA es q^2, la de tener Aa es $2qp$ y la de tener aa es p^2. Lo mismo aplicaría para BB, Bb, y bb, pues estamos asociando la misma MAF a ambos *loci*. En la Tabla 2.3 se presentan las probabilidades que resultan para cada combinación de alelos aplicando esta operación.

Para calcular la heredabilidad máxima para un valor de prevalencia conocido —recordemos, 0,8—, tenemos que usar la Ecuación 2.6. Tal y como quedó demostrado en la Ecuación 2.8,

	BB	Bb	bb
AA	0,71334	0,77620	0,84459
Aa	0,77620	0,84459	0,91902
aa	0,84459	0,91902	1

Tabla 2.4: Tabla de penetrancia resultante para el ejemplo numérico de la sección 2.3.1, calculada manualmente

	BB	Bb	bb
AA	0,71333708	0,77619546	0,84459285
Aa	0,77619546	0,84459285	0,91901733
aa	0,84459285	0,91901733	1,00000000

Tabla 2.5: Tabla de penetrancia resultante para el ejemplo numérico de la sección 2.3.1, calculada por Toxo

$x(1+y)^4$ es la mayor de las expresiones del modelo para valores en $\mathbb{R}+$, por lo que será la que empleemos para igualarla a la unidad y expresar así la maximización de la heredabilidad en el segundo miembro del sistema resultante.

Para el primer miembro del sistema, simplemente hemos de reemplazar $P(D|g_i)$ por las expresiones del modelo (véase Tabla 2.2) y $P(g_i)$ por cada una de las probabilidades calculadas para cada genotipo.

Así, pues, resulta el sistema, tras simplificar:

$$\begin{aligned} \frac{x\ \ y^4+12y^3+54y^2+108y+81\sum}{81} &= 0.8 \\ x(1+y)^4 &= 1 \end{aligned} \tag{2.9}$$

Las soluciones a este sistema de ecuaciones, considerando las restricciones que aplican, son $x = 0,71334$ e $y = 0,08812$. La heredabilidad resultante es 0,02600 y la prevalencia es 0,8. En la Tabla 2.4, presentamos la tabla de penetrancia final que obtendríamos con las soluciones calculadas y que pone fin a este proceso.

Por último, procedemos a contrastar los cálculos anteriores, realizados manualmente, con los resultados que ofrece Toxo para la configuración de este ejemplo. Las soluciones que encuentra Toxo para el sistema de ecuaciones son $x = 0,71333708$ e $y = 0,08811877$. La heredabilidad resultante es 0,02600183 y la prevalencia es 0,80000000. En la Tabla 2.5, presentamos la versión de la Tabla 2.4 a la que llega Toxo. Como se puede apreciar, todos los resultados son esencialmente los mismos, difiriendo únicamente en la cantidad de decimales empleados.

Capítulo 3

PyToxo: una herramienta Python para calcular tablas de penetrancia de modelos de epistasia de alto orden

En este capítulo introducimos PyToxo. Comenzamos por justificar la motivación de su creación a partir de las limitaciones de Toxo, para luego centrarnos en el diseño *software* y en el funcionamiento de la nueva versión en Python.

El código fuente de PyToxo está liberado para uso de la comunidad científica y puede consultarse en el repositorio GitHub del proyecto [1].

3.1 Introducción

En el capítulo anterior afrontamos diferentes métodos matemáticos para el cálculo de tablas de penetrancia, deteniéndonos en especial en el método utilizado por la librería Toxo [2], desarrollada por Ponte-Fernández *et al.* [21]. Si bien esta librería aporta una serie de innovaciones interesantes al estado del arte, presenta también sus propias limitaciones, sobre todo en materia de usabilidad. En estas carencias es exactamente donde se halla la motivación para llevar a acabo este Trabajo de Fin de Grado.

Como expusimos someramente en el Capítulo 1, la restricción más evidente que presenta Toxo para un usuario final es que, al estar implementado en MATLAB [46], requiere que el usuario disponga del mismo para trabajar con él. MATLAB, de MathWorks, es un *software* comercial de pago. Si bien también ofrece un entorno disponible sin licencia en el que se

[1] Código fuente disponible en `https://github.com/bglezseoane/pytoxo` [6]
[2] Código fuente disponible en `https://github.com/UDC-GAC/toxo` [20]

pueden ejecutar programas sin necesidad de instalar el propio *software* completo, como Toxo es una librería y no un programa completo con una interfaz de usuario, no es posible emplear esta alternativa. Así pues, el usuario necesitaría una licencia de MATLAB para poder ejecutar Toxo.

Además, Toxo requiere que el usuario disponga de conocimientos relativamente avanzados para utilizarlo. La librería no dispone de interfaz como tal y el usuario ha de manipular directamente el código del programa, llegando incluso a tener que modificarlo para lograr ciertos ajustes. Es por ello que mejorar la usabilidad de la herramienta es un objetivo clave para favorecer su uso, como de hecho se menciona al final del artículo de Ponte-Fernández *et al.* [21].

También es mejorable la precisión numérica del cálculo de las tablas de penetrancia que manifiesta Toxo, permitiendo así emplear tablas de penetrancia con expresiones matemáticas más complejas.

Finalmente, siempre es deseable disponer de algún medio de distribución que permita al usuario una instalación sencilla y asegure las dependencias, como el que provee un gestor de paquetes.

Figura 3.1: Anagrama de PyToxo

Como avanzábamos en el Capítulo 1, PyToxo es una nueva librería implementada en Python para el cálculo de tablas de penetrancia de cualquier orden, basada en la aproximación matemática de Toxo y mejorando sus flaquezas identificadas. En la Figura 3.1 mostramos el logotipo elaborado para la susodicha librería.

El diseño de PyToxo y la elección de las librerías sobre las que sustentará su funcionamiento han de permitirle cubrir toda la funcionalidad que a día de hoy soporta Toxo, sobre todo en el aspecto matemático, de cara a la resolución de los casos que es capaz de manejar.

PyToxo ha de contar con una interfaz programática en forma de librería —para poder ser utilizada en otros programas Python fácilmente—, una interfaz en línea de comando —para usuarios avanzados o para poder ejecutar el programa en modo de procesamiento por lotes (*batch*)— y una interfaz gráfica —especialmente orientada para usuarios poco familiarizados con entornos de ejecución en línea de comandos—. La elección de tecnologías de PyToxo ha de tener especialmente en cuenta todos estos requisitos.

3.1.1 Requisitos formales de PyToxo

A continuación listamos los requisitos de grano grueso identificados para PyToxo, para terminar de definir el propósito de este *software*:

1. Reimplementación de la librería Toxo en Python, conservando la plena capacidad operativa y matemática de la versión en MATLAB.

2. Ampliación de los modelos epistáticos soportados por Toxo, de forma que PyToxo resuelva todos los casos que resuelve Toxo y algunos más.

3. Mejora de la precisión numérica en las tablas de penetrancia resultado.

4. Implementación de una interfaz programática para que PyToxo pueda ser utilizado cómodamente como librería desde otros programas Python o desde una sesión interactiva del mismo.

5. Implementación de una interfaz en línea de comando para que PyToxo pueda ser utilizado cómodamente desde el entorno de ejecución en línea de comandos de un terminal POSIX de sistemas tipo Unix o un CMD de Windows.

6. Implementación de una interfaz gráfica (GUI) para que PyToxo pueda ser utilizado cómodamente desde un entorno de ventanas de escritorio compatible con las plataformas Linux, Mac OS y Windows en sus versiones relativamente modernas.

7. Incorporación de PyToxo al repositorio PyPI para que pueda ser instalado fácilmente en cualquier máquina.

3.2 Elección de Python como lenguaje de programación

Como ya se ha evidenciado en multitud de ocasiones en el presente texto, e incluso en el propio nombre del *software*, el lenguaje de programación escogido para la nueva implementación ha sido Python. No obstante, hasta este momento no habíamos justificado todavía esta decisión. Procedemos a hacerlo precisamente en esta sección.

Python [30] es un lenguaje de programación interpretado, dinámico, multiparadigma y multiplataforma que destaca por su código fácilmente legible y limpio. Soporta orientación a objetos, programación imperativa y, parcialmente, programación funcional. Es administrado por la Python Software Foundation y posee una licencia de código abierto de redacción propia.

Hoy en día Python es uno de los lenguajes de programación más ampliamente utilizados en el mundo [2, 19, 40, 49]. Su sintaxis simple y de alto nivel permite armar prototipos en muy

poco tiempo y el gran número de librerías de calidad con las que cuenta hacen que se pueda emplear con garantías para casi cualquier tipo de proyecto.

Algunas de las librerías de más renombre de Python son aquellas que lo hacen especialmente atractivo para la comunidad científica: NumPy [17], para computación numérica; SciPy [37], para computación científica en general; SymPy [41], para matemática simbólica; etc.

Las características de este lenguaje lo hacen muy atractivo para profesionales de ámbitos relativamente lejanos a la programación, debido a su poco pronunciada curva de aprendizaje. Así, es una de las opciones más recurridas por los biólogos y los profesionales de las ciencias de la salud que comienzan a trabajar en el campo interdisciplinar de la bioinformática, como evidencia el proyecto Biopython [3] o como se expone en Mariano *et al.* [14]: «Nuestros resultados también señalan la tendencia reciente de utilizar Python como lenguaje de programación para aplicaciones bioinformáticas».

Además de todo lo anterior, Python también soporta la implementación de interfaces gráficas, llegando a disponer de una librería para ello en la distribución de serie del lenguaje.

En último lugar, Python cuenta con PyPI [29], un repositorio de *software* oficial para paquetes de terceros. PyPI es un catálogo cuantioso de aplicaciones desarrolladas en el lenguaje que permite una distribución sencilla y bien conocida por los usuarios del mismo.

Así pues, una vez considerado todo lo anterior y los objetivos planteados para la herramienta PyToxo, la elección de Python como lenguaje para su desarrollo nos parece bien fundada.

PyToxo es una herramienta enfocada al ámbito de la bioinformática, concretamente a aplicaciones de genética. El hecho de estar desarrollada en uno de los lenguajes más populares entre la comunidad profesional que le daría uso a la librería es un poderoso argumento para su elección. Más si cabe, si este lenguaje también es, como ya hemos expuesto, uno de los más utilizados del mundo en general.

La posibilidad de utilizar en el desarrollo de la herramienta librerías de tanta calidad, tan populares y tan bien mantenidas como NumPy o SymPy también se ha valorado como un punto a favor, pues estos recursos pueden simplificar notablemente la implementación del proyecto.

Dado que uno de los requerimientos de PyToxo es la elaboración de una interfaz gráfica para la librería, se ha tenido en cuenta que el lenguaje escogido para el proyecto permita su implementación, sin tener que recurrir a terceras tecnologías. Como ya se ha mencionado, Python lo hace.

Por último, pero no menos importante, ha tenido su peso la posibilidad de distribuir la herramienta por medio de PyPI, permitiendo así que los potenciales usuarios instalen PyToxo fácilmente.

Para concluir este análisis, tenemos que valorar también las desventajas que presenta Python, pues como todos los lenguajes de programación existentes, presenta puntos fuertes y puntos débiles. En el caso de Python, un lenguaje interpretado, su mayor debilidad es la eficiencia. Python no puede competir con lenguajes compilados —como por ejemplo, C [45]— en este aspecto.

Aunque PyToxo no es un programa que implique computación intensiva, sí es cierto que para sus cálculos termina manejando sistemas de ecuaciones relativamente grandes, sobre todo si estamos trabajando con modelos epistáticos de orden alto. Sin embargo, en este momento, tal y como se verá en el Capítulo 5, el tiempo de ejecución no supone un problema, por lo que se ha descartado optar por otro lenguaje de programación diferente a Python. El implementar esta librería en C o C++, que son lenguajes que a priori se esperan mucho más eficientes, requeriría un esfuerzo mucho mayor, por tratarse también de lenguajes de más bajo nivel de abstracción. Además, la usabilidad de la librería se podría ver comprometida, pues como hemos expuesto, Python es el lenguaje más utilizado por la comunidad objetivo de PyToxo.

3.3 Entorno de desarrollo

PyToxo se ha desarrollado en y para Python 3.8. La versión 3.8 es una de las más modernas del lenguaje —actualmente la más moderna es la 3.9— y a la vez está lo suficientemente madura como para estar presente de serie en la mayoría de entornos y sistemas operativos del mercado. La última versión de Python 3.8, Python 3.8.10, se publicó el 3 de mayo de 2021 y se puede descargar desde el sitio web oficial de Python [31]. Según el calendario de los desarrolladores, esta es la versión final de mantenimiento para Python 3.8.

Además del propio lenguaje, se han empleado una serie de librerías para implementar PyToxo. Las listamos a continuación:

1. *mpmath* [10] 1.2.1: es una de las librerías que emplea internamente SymPy. Desde PyToxo se utiliza sólamente para acceder a algunos parámetros de configuración y ajustar el resolutor de SymPy.

2. NumPy [17] 1.20.3: librería para computación numérica por excelencia de Python. En PyToxo no se utiliza internamente, sino sólo para permitir al usuario trabajar con datos en formato NumPy, para favorecer la integración.

3. Pillow [12] 8.2.0: librería para el tratamiento de imágenes. Se usa para algunos detalles de la interfaz gráfica.

4. PySimpleGUI [23] 4.43.0: librería que simplifica la implementación de interfaces gráficas en Python. Es la dependencia principal de la interfaz gráfica de PyToxo.

5. SymPy [41] 1.8: librería de matemática simbólica de Python por antonomasia. Se utiliza en la práctica totalidad del flujo numérico de PyToxo, para representar los datos numéricos y realizar las operaciones necesarias.

6. Termcolor [11] 1.1.0 y Colorama [7] 0.4.4: facilitan el uso de colores en la salida de una interfaz en línea de comando para mejorar la legibilidad.

Para la interfaz gráfica, pese a implementarla a través de la librería PySimpleGUI, esta se termina apoyando en Tk [32], que viene instalada de serie con la mayoría de las distribuciones de Python. No obstante, no siempre es así en algunas instalaciones de Linux. En estos casos PyToxo detecta la carencia y le sugiere al usuario instrucciones para repararla al instante.

Además de lo anterior, durante la fase de desarrollo se han utilizado algunas dependencias adicionales que también serían necesarias para ejecutar las pruebas automáticas, si bien no se especifican como dependencias en una instalación ordinaria para un usuario final. Estas dependencias adicionales son: Black 21.5b2 y sus propias dependencias, GitPython 3.1.17, PSUtil 5.8.0, Setuptools 47.1.0 y Tabulate 0.8.9.

Setuptools [25] es a día de hoy el conjunto de herramientas recomendado por los desarrolladores del lenguaje Python para preparar la distribución de *software* a través de PyPI. Black [53] es una herramienta que permite estilizar el formato del código fuente Python siguiendo una convención estándar, lo que lo hace de mejor calidad y más legible por toda la comunidad de desarrolladores.

GitPython, PSUtil y Tabulate sólo se usan para la composición automática de informes durante el proceso de pruebas automáticas. Se puede encontrar información sobre todas ellas consultando el repositorio PyPI.

Como herramientas para el desarrollo, se han utilizado el IDE JetBrains PyCharm 2020.3.4 Professional [9], del cual la Universidade da Coruña provee de licencia, y Git [39] para el control de versiones.

3.4 Diseño *software* de PyToxo

En esta sección describiremos los diferentes componentes que integran PyToxo a nivel de *software*.

En este capítulo, y en especial en esta sección, intentamos ser precisos a la hora de referirnos a los diferentes entes *software* que conforman PyToxo. Así, cuando decimos *paquete* no hacemos referencia a una noción abstracta, sino que apuntamos directamente a un paquete Python; cuando decimos *módulo*, nos referimos a un módulo Python; etc.

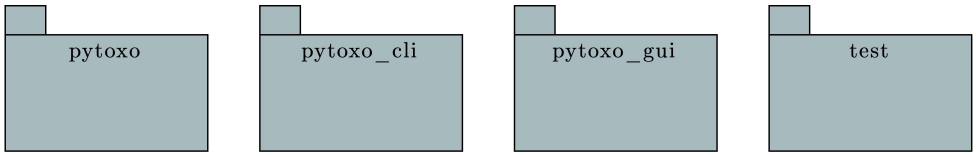

Figura 3.2: Diagrama UML de paquetes del repositorio de PyToxo

3.4.1 Diseño a nivel de paquetes y módulos

Tal y como sintetizamos en el diagrama de la Figura 3.2, en el repositorio de PyToxo [6] nos encontramos el código fuente organizado en cuatro paquetes Python diferentes:

- `pytoxo`: el paquete central del repositorio, que contiene toda la lógica interna de la librería y la interfaz programática para poder ser invocado desde otros programas Python.
- `pytoxo_cli`: contiene la interfaz en línea de comando, independizándola de la librería en sí.
- `pytoxo_gui`: contiene la interfaz gráfica de usuario, independizándola de la librería en sí.
- `test`: contiene todas las pruebas automáticas del repositorio.

Así, desde un primer momento se aísla la lógica de la librería de las interfaces proporcionadas, que serían modificables o incluso sustituibles sin que la primera tuviera ninguna afectación. También se podrían distribuir de forma independiente, pues para Python son paquetes completamente diferenciados. `pytoxo_cli` es un paquete del que `pytoxo` es simple dependencia, al igual que `numpy` lo es de `pytoxo`.

El paquete `pytoxo`

El paquete `pytoxo` contiene toda la lógica de la librería PyToxo y es el único necesario para poder utilizar la librería desde otros programas Python. El paquete está estructurado en cinco módulos diferentes, que describimos a continuación.

- `model`: contiene la clase `Model`, que abordaremos más adelante.
- `ptable`: contiene la clase `PTable`, que abordaremos más adelante.
- `calculations`: contiene funciones basadas en cálculos matemáticos para reusar desde diferentes partes de la implementación, como por ejemplo para computar la prevalencia dada una lista de penetrancias.

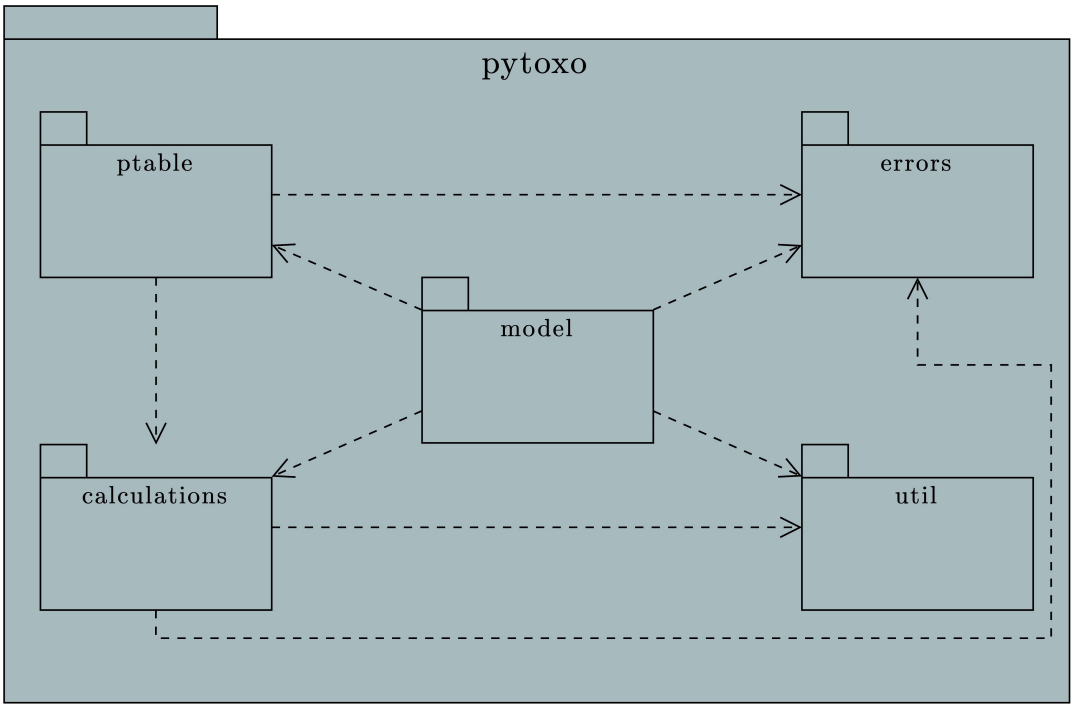

Figura 3.3: Diagrama UML de módulos del paquete `pytoxo`

- `util`: contiene funciones de utilidad más genéricas que se emplean en la implementación.

- `errors`: contiene las definiciones de las excepciones *ad hoc* de PyToxo.

En el diagrama de la Figura 3.3, podemos apreciar las dependencias que existen entre los módulos. `model` hace uso de todos los demás módulos, `ptable` necesita de `calculations` y `errors`, mientras que `calculations` precisa de `util` y `errors`.

Los paquetes `pytoxo_cli` y `pytoxo_gui`

Los paquetes `pytoxo_cli` y `pytoxo_gui` contienen respectivamente la interfaz en línea de comando y la interfaz gráfica de PyToxo. Se han concebido como paquetes separados para posibilitar así su desarrollo y distribución independientes de la librería como tal.

Ambos contienen un único módulo principal que hace de punto de entrada al programa para cada uno de sus casos de uso asociados: el uso de PyToxo desde un entorno de ejecución en línea de comando y el uso de PyToxo desde un entorno gráfico de ventanas.

Ambos dependen del paquete `pytoxo` para ofrecer alguna funcionalidad.

El paquete `test`

El paquete `test` contiene todas las pruebas automáticas de PyToxo, que están divididas en cuatro subpaquetes: `unit`, `integration`, `solubility` y `accuracy`.

El subpaquete `unit` contiene pruebas unitarias que se centran en comprobar el correcto funcionamiento de partes concretas de la implementación, como las funciones de tipo matemático del paquete `calculations`, la correcta generación de objetos `Model` a partir de diferentes entradas, la composición del archivo de salida para las tablas de penetrancia calculadas, etc.

El subpaquete `integration`, por su parte, contiene pruebas de integración en las que se analiza el correcto funcionamiento del código cuando se implican varios módulos o partes separables de la implementación, comprobando así que la comunicación entre todas ellas se realiza correctamente y que el flujo del programa transcurre como debe. En estas pruebas se verifica, entre otras cosas, el proceso completo del cálculo de las soluciones de una tabla de penetrancia, desde la generación del objeto `Model` hasta obtener la salida de la tabla deseada. También se prueba que las excepciones son debidamente lanzadas e interceptadas en diversos puntos.

El subpaquete `solubility` tiene el cometido de probar que PyToxo es capaz de resolver al menos todo lo que resuelve Toxo, y algunos casos más.

Por último, el subpaquete `accuracy` contiene pruebas para aseverar que las soluciones matemáticas de PyToxo se hallan dentro de los márgenes de precisión deseados.

Para varias de las comprobaciones anteriores, estas pruebas comparan el funcionamiento de PyToxo con el de Toxo. Para ello, en el repositorio se han recopilado una colección de archivos de tablas calculadas con Toxo, así como otros registros, que estos test interpretan para cursar su cometido.

Por otra parte, los módulos `solubility` y `accuracy` también tienen implementado el comportamiento opcional de generar informes acerca de los resultados que obtienen, útiles para incorporar por ejemplo al presente documento.

Para la implementación de las pruebas, se ha empleado fundamentalmente el *unit testing framework* (`unittest`) de Python [34], que se distribuye de serie con el lenguaje y tiene flexibilidad suficiente para poder emplearse tanto para las pruebas unitarias, como para las de integración, y en el caso de PyToxo las específicas de solubilidad y precisión.

En total, toda la rutina de pruebas automáticas de PyToxo conlleva la ejecución de 111 test, que tardan en ejecutarse entre 1,5 y 2 horas, y tienen un 80 % de cobertura del código fuente.

Para la primera versión distribuible de PyToxo, todas las pruebas han sido superadas correctamente en Linux, Mac OS y Windows.

3.4.2 Diseño a nivel de clases

A nivel de clases, PyToxo se articula en dos clases fundamentales: `Model` y `PTable`, como bien reflejamos en el diagrama UML de la Figura 3.4.

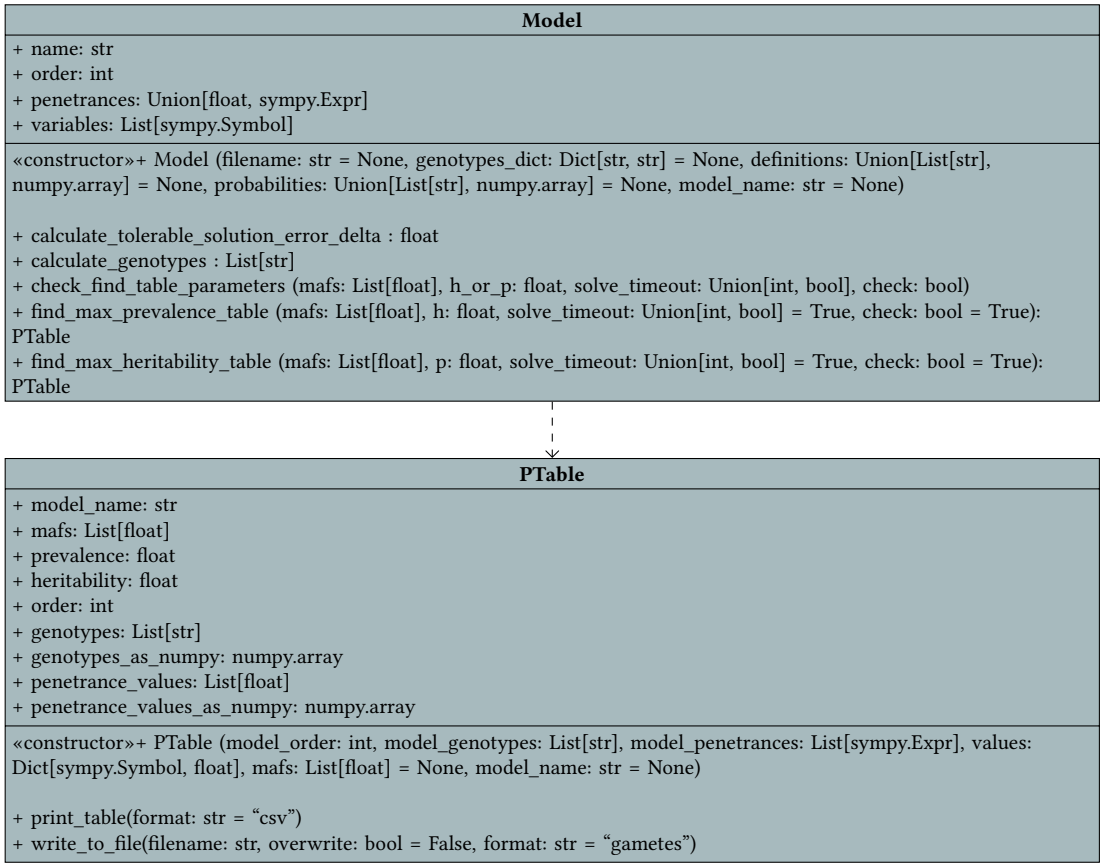

Figura 3.4: Diagrama UML de clases principales del paquete `pytoxo`, ocultando atributos y métodos privados

La clase `Model` es el pivote del flujo de la librería. Representa un modelo epistático de los que maneja PyToxo, además de los datos con los que el usuario quiere proceder a calcular la tabla. Así, asociado a un `Model` tendríamos un nombre arbitrario para que el usuario lo indentifique (`name`), el orden del modelo (`order`), la relación de penetrancias del modelo (`penetrances`) y los nombres de las variables que se emplean para la definición de esas penetrancias (`variables`).

Para implementar el constructor de `Model` se ha emulado el principio de la sobrecarga (*overloading*, en inglés), que en Python no está soportado como tal. Así, en función de los parámetros opcionales que el usuario emplee al llamar al constructor, este terminará siendo invocado de una de las tres formas que se listan a continuación:

1. Generando el modelo simplemente a partir de la ruta del archivo CSV que lo contiene.
2. Generando el modelo a partir de un diccionario Python con las definiciones del genotipo y sus probabilidades asociadas.
3. Generando el modelo a partir de dos listas o *arrays* de Numpy, una con las definiciones del genotipo y otra con sus probabilidades asociadas. También es posible utilizar una lista y un *array* de Numpy combinados, o viceversa.

Gracias a esta polivalencia, el constructor resulta casi inmediato para ser utilizado tanto para satisfacer una interfaz como para su uso directo por parte de usuarios programadores.

Como métodos públicos, en `Model` se exponen:

1. `calculate_tolerable_solution_error_delta`: halla el margen de error tolerable para el cálculo de las soluciones del sistema cuando se calcula una tabla de penetrancia, basándose en el orden del modelo con el que se está trabajando.
2. `calculate_genotypes`: calcula la lista con las definiciones de genotipo que le corresponden el modelo, dado su orden.
3. `check_find_table_parameters`: comprueba si una serie de parámetros dados como argumentos serían válidos para proceder con el cálculo de una tabla de penetrancia con los métodos que siguen.
4. `find_max_prevalence_table`: calcula una tabla de penetrancia maximizando la prevalencia, a partir del modelo y los parámetros dados como argumentos.
5. `find_max_heritability_table`: calcula una tabla de penetrancia maximizando la heredabilidad, a partir del modelo y los parámetros dados como argumentos.

Los dos métodos más relevantes de la clase `Model` son `find_max_prevalence_table` y `find_max_heritability_table`. Cualquiera de ellos, en caso de que el proceso resulte exitoso, devuelve una tabla de penetrancia como un objeto de la clase `PTable`. Esta clase está concebida como un contenedor para la tabla resultado, permitiendo que un usuario programador la pueda manipular cómodamente y que se pueda explotar de manera muy directa desde una interfaz u otro programa que haga uso de ella.

A pesar de ser accesorios, los métodos `calculate_tolerable_solution_error_delta`, `calculate_genotypes` y `check_find_table_parameters` son públicos porque resultan de utilidad a la hora de implementar terceros módulos que hagan uso de esta clase, como por ejemplos las interfaces propias de PyToxo.

Por su parte, la clase `PTable` contiene como atributos el nombre del modelo al que está asociada la tabla (`model_name`), así como su orden (`order`); las MAF (`mafs`), la prevalencia

(`prevalence`) y la heredabilidad (`heritability`) empleadas en su cálculo; las definiciones de genotipo (`genotypes`); las definiciones de genotipo en formato propio de Numpy (`genotypes_as_numpy`); las penetrancias resultantes (`penetrance_values`); y las penetrancias resultantes en formato Numpy (`penetrance_values_as_numpy`).

Como métodos, `PTable` cuenta con dos operaciones asociadas para permitir imprimir en pantalla o guardar en un archivo la tabla generada. Para ambos casos, los formatos soportados son un simple archivo CSV y el formato empleado por GAMETES [50]. Con este último las tablas obtenidas por PyToxo pueden pasarse directamente al programa de Urbanowicz *et al.* [50] para simular un conjunto de datos de una población que manifiesta la interacción epistática descrita por la tabla de penetrancia proporcionada.

Además de `Model` y `PTable`, el paquete `pytoxo` contiene una serie de clases que representan las diferentes excepciones que se pueden dar durante su funcionamiento. Las exponemos en el diagrama de la Figura 3.5.

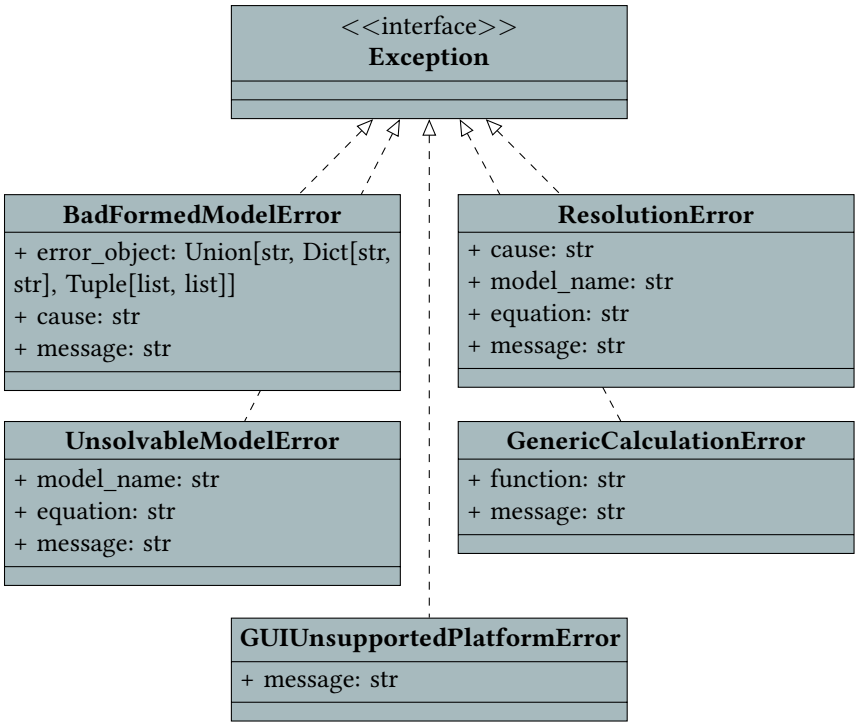

Figura 3.5: Diagrama UML de clases de excepciones del paquete `pytoxo`

Las diferentes excepciones recogen las siguientes casuísticas:

- `BadFormedModelError`: error durante el procesamiento de un archivo CSV, cuando se emplea para intentar generar un objeto `Model`.
- `ResolutionError`: error de resolución del sistema de ecuaciones al hallar una tabla de

penetrancia.

- `UnsolvableModelError`: el sistema de ecuaciones compuesto para la generación de una tabla de penetrancia es irresoluble.
- `GUIUnsupportedPlatformError`: error de compatibilidad con la plataforma al tratar de ejecutar la interfaz gráfica.
- `GenericCalculationError`: error anormal no identificado en las operaciones del módulo `calculations`.

Como quedó evidenciado en el diagrama de la Figura 3.3, todos los demás módulos hacen uso del módulo `errors`.

3.5 Cálculo de las tablas de penetrancia

En la sección anterior expusimos cómo los métodos `find_max_prevalence_table` y `find_max_heritability_table`, de la clase `Model`, son los responsables de iniciar el proceso del cálculo de una tabla de penetrancia, una vez que el usuario ha cumplimentado toda la información necesaria a través de la creación de ese objeto `Model`.

En esta sección abordaremos lo que ocurre una vez son invocados estos métodos, desde la perspectiva de la algoritmia, pues es en este punto donde radica la mayor parte de la carga computacional del flujo del programa.

A continuación presentamos los pseudocódigos 1 y 2, en los que desglosamos de forma abstracta ambos métodos.

Entrada: `mafs`, `h`, `timeout`, `check`

1: Validar argumentos con `check_find_table_parameters`
2:
3: Construir sistema de ecuaciones, `eq_system`, a partir de `mafs` y `h`
4: `sol = _solve(eq_system, timeout)` // Método privado de la clase `Model`
5:
6: **if** `check` **then**
7: `_check_solution(eq_system, sol)` // Método privado de la clase `Model`
8: **end if**
9:
10: Usar constructor `PTable` para generar `ptable` a partir de la solución `sol` y los datos del modelo actual

Salida: `ptable`

Pseudocódigo 1: `Model.find_max_prevalence_table`

Entrada: `mafs`, `p`, `timeout`, `check`

1: Validar argumentos con `check_find_table_parameters`
2:
3: Construir sistema de ecuaciones, `eq_system`, a partir de `mafs` y `p`
4: `sol = _solve(eq_system, timeout)` // Método privado de la clase `Model`
5:
6: **if** `check` **then**
7: `_check_solution(eq_system, sol)` // Método privado de la clase `Model`
8: **end if**
9:
10: Usar constructor `PTable` para generar `ptable` a partir de la solución `sol` y los datos del modelo actual

Salida: `ptable`

Pseudocódigo 2: `Model.find_max_heritability_table`

Como se puede apreciar en los anteriores pseudocódigos, ambos métodos responden a una estructura común. Difieren en los datos que reciben como entrada: la prevalencia o la heredabilidad, dependiendo de lo que se fije y de lo que se maximice. Luego se hace uso del mismo método de validación de argumentos, `check_find_table_parameters`, para ambos casos.

El siguiente paso para cada uno de los dos métodos es la construcción del sistema de ecuaciones a resolver, a partir de las ecuaciones 2.4 o 2.6, expuestas en el Capítulo 2. Una vez que el sistema ha sido generado, ambos métodos confluyen en el método privado `_solve`, que contiene la estrategia de resolución de PyToxo. Lo analizaremos más adelante.

Una vez calculadas las soluciones para x e y en el sistema, se verifica que esta solución sea correcta dadas las restricciones vigentes, siempre y cuando el usuario no haya desactivado este comportamiento, que es opcional.

Por último, se utiliza `sol` y otros datos correspondientes al objeto `Model` para generar una tabla de penetrancia como objeto `PTable`, que resulta la salida de ambos métodos `find_max_prevalence_table` y `find_max_heritability_table` y que concluye el proceso de cálculo.

Volviendo atrás en la explicación, es necesario afrontar el método privado `Model._solve`. Este método asume la responsabilidad de resolver el sistema de ecuaciones conformado por sus llamadores. Además, recibe un segundo argumento del que hasta ahora no habíamos ofrecido demasiada información, `solve_timeout`. Este parámetro puede tomar valores *booleanos* o enteros y sirve para controlar el tiempo máximo de espera que se le asocia al intento de resolución del sistema de ecuaciones. Si el usuario lo fija como falso, no se empleará ningún tiempo máximo; si ingresa un entero, se interpretará como el tiempo máximo, en segundos; y si se marca como verdadero, el tiempo máximo será asumido heurísticamente a partir del

orden del modelo, basándose en la Ecuación 3.1. Si el tiempo máximo de espera es excedido, se aborta el proceso y se lanza una excepción informando de ello.

$$solve\_timeout = 60(order + 1)^2 \tag{3.1}$$

El hecho de que se emplee este tiempo máximo radica en que el resolutor de la librería SymPy [41], último responsable del proceso de solución matemática del sistema, puede no converger para ciertas configuraciones. En estos casos, es bastante habitual que se dilate mucho en el tiempo en su intento, aunque con la experiencia se ha observado que cuando no lo consigue en un cierto margen de tiempo, termina por no conseguirlo. Introduciendo este parámetro se puede mejorar la experiencia del usuario abortando procesos que muy probablemente resulten infructíferos.

Cabe mencionar que, de entre los varios resolutores para sistemas de ecuaciones que ofrece SymPy [42], se ha optado finalmente por `sympy.solve`, a pesar de que en un primer momento el candidato ideal parecía ser `sympy.nonlinsolve`. Esta decisión se basó simplemente en el desempeño logrado con cada una de las opciones posibles.

Otro de los parches que se han implementado alrededor del resolutor `sympy.solve` es el que atañe a la precisión de sus cálculos internos. SymPy emplea internamente la librería *mpmath* [10], la cual tiene asociado un parámetro de precisión que controla el error acumulado que se considera tolerable. Para el caso de PyToxo, sobre todo cuando se trabaja con modelos grandes, resulta posible relajar ese margen de error acumulable para tratar de converger a una solución del sistema. Así pues, cada vez que se inicia `Model._solve`, se intenta inicialmente resolver el sistema de ecuaciones sin alterar ninguna configuración y, sólo si no resulta posible, es cuando se incrementa el error acumulable y se vuelve a intentar.

No obstante, PyToxo tiene su propia política de precisión para la solución de la tabla de penetrancia, que es plenamente respetada por el anterior ajuste en todos los casos, y que se hace valer al aplicar el método `Model._check_solution` sobre los valores calculados, como hemos expuesto en los fragmentos de pseudocódigo 1 y 2.

Para concluir esta explicación, en el Pseudocódigo 3 presentamos una abstracción del método `Model._solve` que acabamos de tratar.

Por último, en toda esta sección mantenemos una simplificación en cuanto a las explicaciones relativas al tratamiento de los casos de excepción. En el diagrama de la Figura 3.5 hemos representado las diferentes clases que representan los posibles errores que se pueden dar durante todo el flujo de ejecución de PyToxo. Como ya hemos explicado en la Sección 3.4.2, algunos de estos errores están vinculados al proceso de cálculo de las tablas de penetrancia y, como tales, parten del método `_solve` para ser interceptados por `Model.find_max_prevalence_table` o `Model.find_max_heritability_table`, y luego por sus llamadores, de forma que finalmente el error llega al usuario de la forma más clarificante posible.

Entrada: `eq_system`, `timeout`

Resolver `timeout`, dependiendo de si el usuario lo desactiva, lo define o lo delega al método heurístico

`pytoxo.util.timeout(sol = sympy.solve(eq_system))` // Intento 1

if not `sol` **and not** irresoluble **then**
    Relajar margen de error acumulado tolerable de *mpmath*
    `pytoxo.util.timeout(sol = sympy.solve(eq_system))` // Intento 2
    Recuperar margen de error acumulado tolerable de *mpmath* por defecto
end if

if irresoluble **then**
    Declarar irresoluble
end if

Salida: `sol`

Pseudocódigo 3: `Model._solve`

3.6 Distribución y ejemplo de uso

PyToxo está subido a PyPI [29], el repositorio oficial de Python, por lo que su instalación resulta casi instantánea para un usuario final.

Como ya indicamos con anterioridad, PyToxo está desarrollado para funcionar en la versión 3.8 o más moderna de Python. Considerando esta dependencia satisfecha en la máquina en la que se desee utilizar PyToxo, sólo habría que utilizar `pip`, el instalador de paquetes de Python, para instalar PyToxo desde PyPI.

Así pues, se podría instalar PyToxo simplemente con el comando:

```
pip install pytoxo
```

Seguidamente, en esta sección vamos a exponer un ejemplo de uso de PyToxo, empleándolo como librería de Python. En el Código 3.1, comentado pormenorizadamente, presentamos cómo se puede utilizar PyToxo para el cálculo de una tabla de penetrancia para el modelo aditivo de segundo orden de Marchini *et al.* [13]; empleando unas MAF de 0,33 y 0,33; y fijando una prevalencia de 0,8, maximizando la heredabilidad.

El programa de ejemplo anterior concluye mostrando la tabla de penetrancia que presentamos en el Código 3.2. Echando la vista atrás, podemos observar que esta tabla es equivalente a las tablas 2.4 y 2.5 que calculamos en el ejemplo numérico de la Sección 2.3.1, en la que enunciamos la misma configuración para la composición del ejemplo.

En el siguiente capítulo presentaremos las interfaces desde las que se puede hacer uso de PyToxo.

```python
import pytoxo

# Parámetros del ejemplo
modelo_csv = "models/additive_2.csv"  # Ruta al archivo
maf = [0.33, 0.33]
prev = 0.8

# Se crea el modelo a partir del archivo CSV con su definición
modelo = pytoxo.Model(filename=modelo_csv)

# Se genera la tabla de penetrancia inmediatamente
tabla_pen = modelo.find_max_heritability_table(mafs=maf, p=prev)

# Y ya se podría manipular, imprimir o guardar la tabla
tabla_pen.print_table(format="gametes")  # Se elige formato GAMETES
```

Código 3.1: Ejemplo de uso de PyToxo como librería en un programa Python

```
Attribute names:  P0  P1
Minor allele frequencies: 0.330 0.330
x: 0.713337077352276
y: 0.0881187663842681
Prevalence: 0.800000000000000
Heritability: 0.0260018292168110

Table:

0.713337077352276, 0.776195460624718, 0.844592847088037
0.776195460624718, 0.844592847088037, 0.919017326870411
0.844592847088037, 0.919017326870411, 1.00000000000000
```

Código 3.2: Tabla de penetrancia calculada por PyToxo en el Código 3.1

Capítulo 4

Interfaces para PyToxo

Este capítulo expone las diferentes interfaces disponibles para PyToxo, poniendo especial énfasis en la interfaz gráfica de usuario.

Como ya hemos reiterado en los capítulos anteriores, además de poder ser empleado directamente como una librería Python —a lo que nos hemos referido antes como *interfaz programática*—, PyToxo cuenta con dos interfaces diferentes para facilitarle su empleo al usuario: una interfaz en línea de comando o en inglés *command line interface* (CLI) y una interfaz gráfica de usuario o en inglés *graphical user interface* (GUI). Las dos exponen la funcionalidad de la librería base de PyToxo, que es dependencia de ambas. También son independientes entre sí y completamente opcionales a la hora de emplear PyToxo.

4.1 Interfaz en línea de comando

La primera de las interfaces de PyToxo es la interfaz en línea de comando. Desde esta se puede utilizar PyToxo en un terminal de tipo POSIX [8] desde sistemas de tipo Unix, o en un CMD de Windows. Es útil para usuarios avanzados acostumbrados a entornos de ejecución en línea de comandos y también está concebida para permitir que PyToxo pueda ser empleado en modo de procesamiento por lotes (*batch*), lanzando por medio de un *script* varios experimentos secuencialmente.

La CLI puede invocarse simplemente introduciendo el siguiente comando, una vez PyToxo ha sido instalado por medio de las instrucciones de la Sección 3.6:

```
1 pytoxo_cli
```

La interfaz en línea de comandos permite calcular tablas de penetrancia tanto en formato de GAMETES [50] como en CSV. Acepta rutas para los modelos epistáticos a utilizar durante el proceso, así como la especificación del resto de parámetros numéricos: MAF, y prevalencia o heredabilidad, dependiendo de qué parámetro se maximice.

Mediante el comando que mostramos a continuación se puede desplegar la ayuda en línea de la CLI, que detallaremos en el Apéndice A.

```
pytoxo_cli -h
```

En el manual del usuario del Apéndice A también se amplía la información relativa al uso de esta interfaz, pero *grosso modo* avanzamos que responde a la especificación POSIX [8]:

```
pytoxo [-h] [--gametes] (--max_prev | --max_her) <model>
    <prev_or_her> <maf> [<maf> ...]
```

A modo ilustrativo, presentamos a continuación un comando que calcularía una tabla de penetrancia para el modelo de Marchini *et al.* [13] multiplicativo de orden 4, usando como prevalencia 0,6 y como MAF 0,2, 0,3, 0,3 y 0,4. Se maximiza la heredabilidad y se formatea la tabla de penetrancia de salida en el formato de GAMETES.

```
pytoxo_cli multiplicative_4.csv --max_her 0.6 0.2 0.3 0.3 0.4
    --gametes
```

Para la implementación de la CLI, cabe destacar el uso de la librería Argparse [26], incluida en la distribución de serie de Python. Esta librería constituye la recomendación oficial de los desarrolladores de Python para la confección de interfaces en línea de comando, y facilita considerablemente su elaboración, generando por sí sola mensajes de ayuda y asegurando la correcta utilización de los parámetros por parte del usuario final.

4.2 Interfaz gráfica de usuario

Además de la interfaz en línea de comando, presentada en la sección anterior, la distribución de PyToxo también cuenta con una segunda interfaz, la interfaz gráfica de usuario. En este caso se trata de una interfaz basada en ventanas, orientada sobre todo a usuarios menos familiarizados con los entornos de ejecución en línea de comandos o para fines más demostrativos que ganen visibilidad al ser presentados de esta forma.

Habiendo instalado PyToxo por medio de las instrucciones de la Sección 3.6, la GUI puede iniciarse introduciendo el comando que sigue:

```
pytoxo_gui
```

La GUI de PyToxo es plenamente compatible y ha sido probada en los sistemas operativos Linux, Mac OS y Windows. Se basa en una única ventana con una parrilla central que contiene el modelo cargado y sus penetrancias una vez han sido calculadas, alrededor de la cual se sitúan una serie de botoneras que permiten accionar toda la funcionalidad, junto con dos menús contextuales.

La interfaz resulta muy sencilla de utilizar debido a que se adapta dinámicamente al estado en el que se halla el flujo de trabajo. Los botones se van habilitando o deshabilitando en función de si tenemos cargado un modelo, rellenados los campos pertinentes, etc. Por ejemplo, no podemos usar el botón de calcular hasta que todos los parámetros hayan sido cumplimentados, y no podemos rellenar las MAF hasta haber cargado un modelo. A continuación incluimos en la Figura 4.1 un diagrama en el que se describe el flujo de ejecución del programa cuando se emplea la GUI, desde la perspectiva del usuario. Seguidamente, también mostramos una serie de capturas de pantalla que presentan secuencialmente el mismo ejemplo de la sección anterior, esta vez desde la interfaz gráfica.

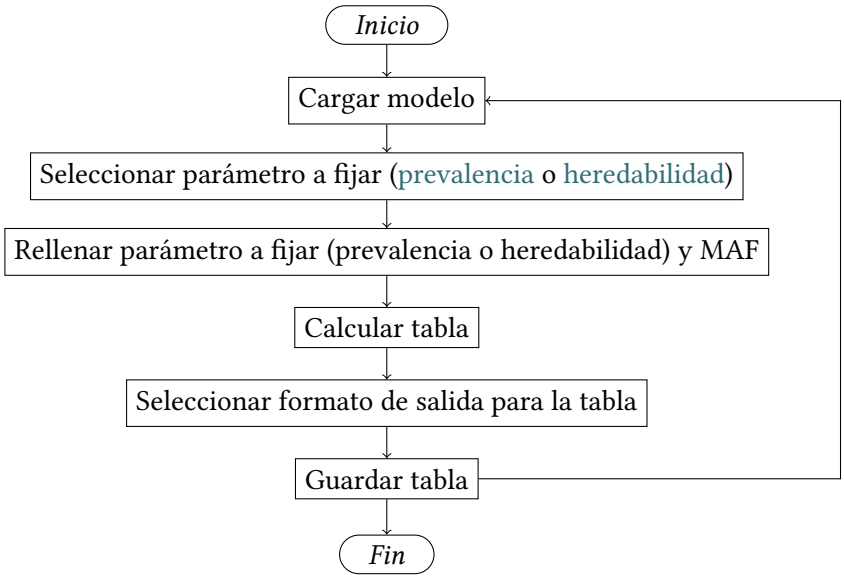

Figura 4.1: Diagrama de flujo del uso básico de PyToxo desde su GUI

En la Figura 4.2 se muestra la pantalla inicial de la interfaz nada más ser abierto el programa. En la Figura 4.3 se aprecia, tras utilizar el menú *file* para seleccionar y cargar un archivo de modelo, cómo este se dispone en la parrilla central y cómo son habilitados diferentes campos del formulario inferior —a destacar el de las cuatro MAF a cumplimentar para el modelo de orden cuatro—. En la Figura 4.4 se muestran rellenados todos los campos anteriores, y vemos también cómo la interfaz muestra algunos indicadores informativos en la parte central, que ayudan al usuario a entender el contexto. Por último, en la Figura 4.5 se lanza el proceso de cálculo de la tabla de penetrancia, habiendo hecho uso del botón *calculate table*, y en la Figura 4.6 vemos el resultado final. A partir de aquí podríamos hacer uso del menú *file* para guardar la tabla como archivo. Vemos en el campo inferior de la interfaz que para esta operación estaría seleccionado el formato de GAMETES de entre los dos disponibles.

Figura 4.2: Pantalla inicial de la GUI de PyToxo desde Mac OS

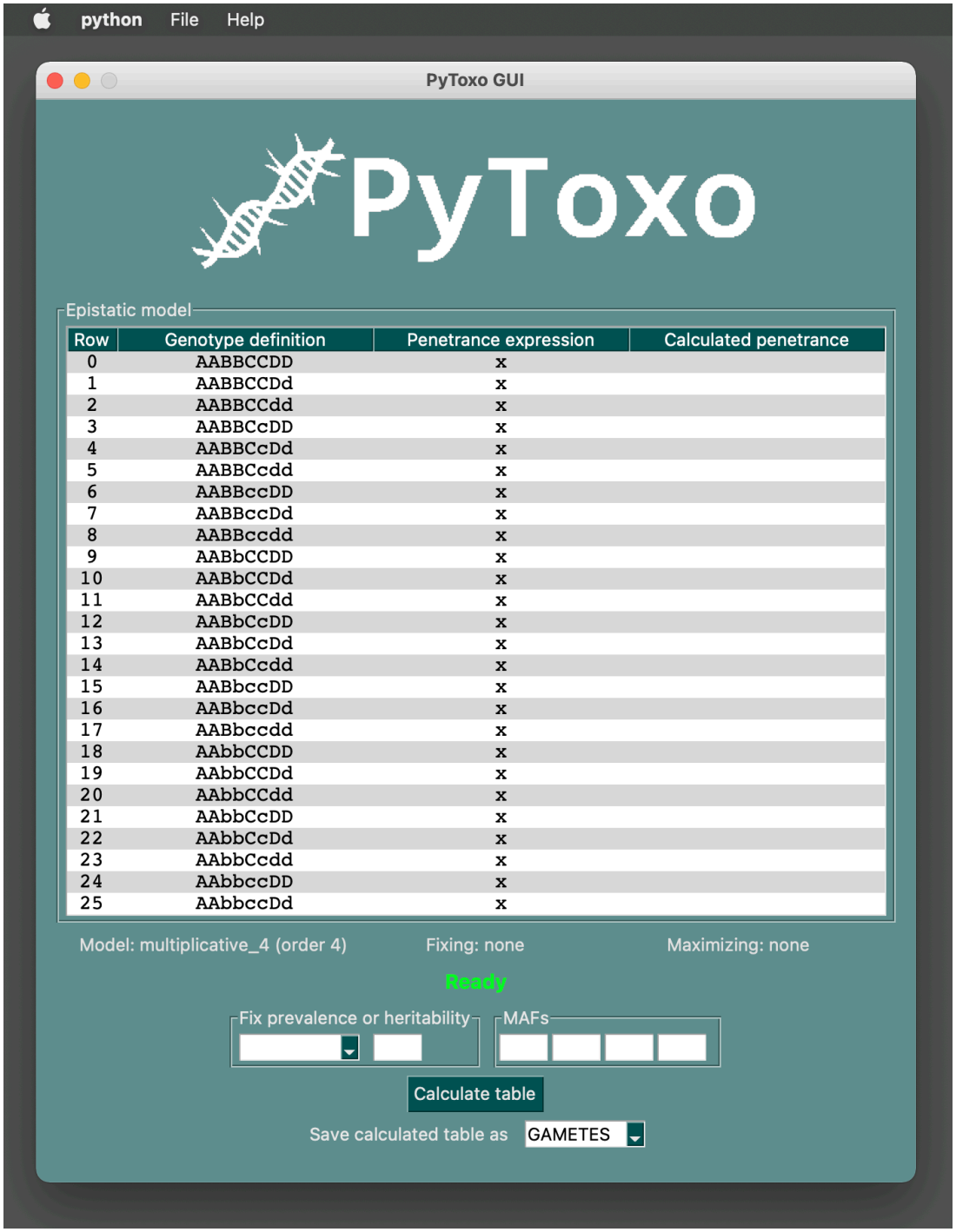

Figura 4.3: Pantalla de la GUI de PyToxo desde Mac OS, tras cargar un modelo

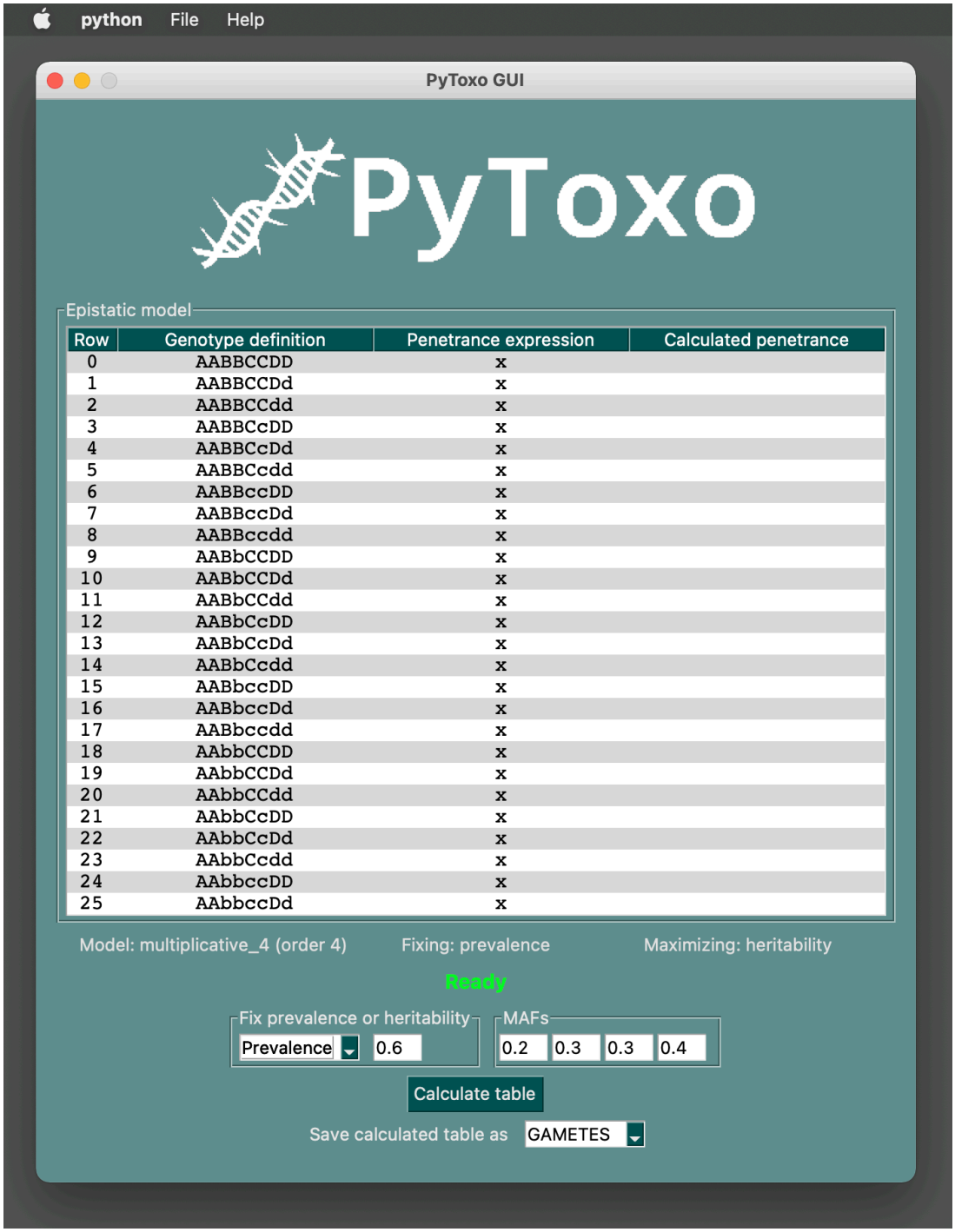

Figura 4.4: Pantalla de la GUI de PyToxo desde Mac OS, con los campos rellenados

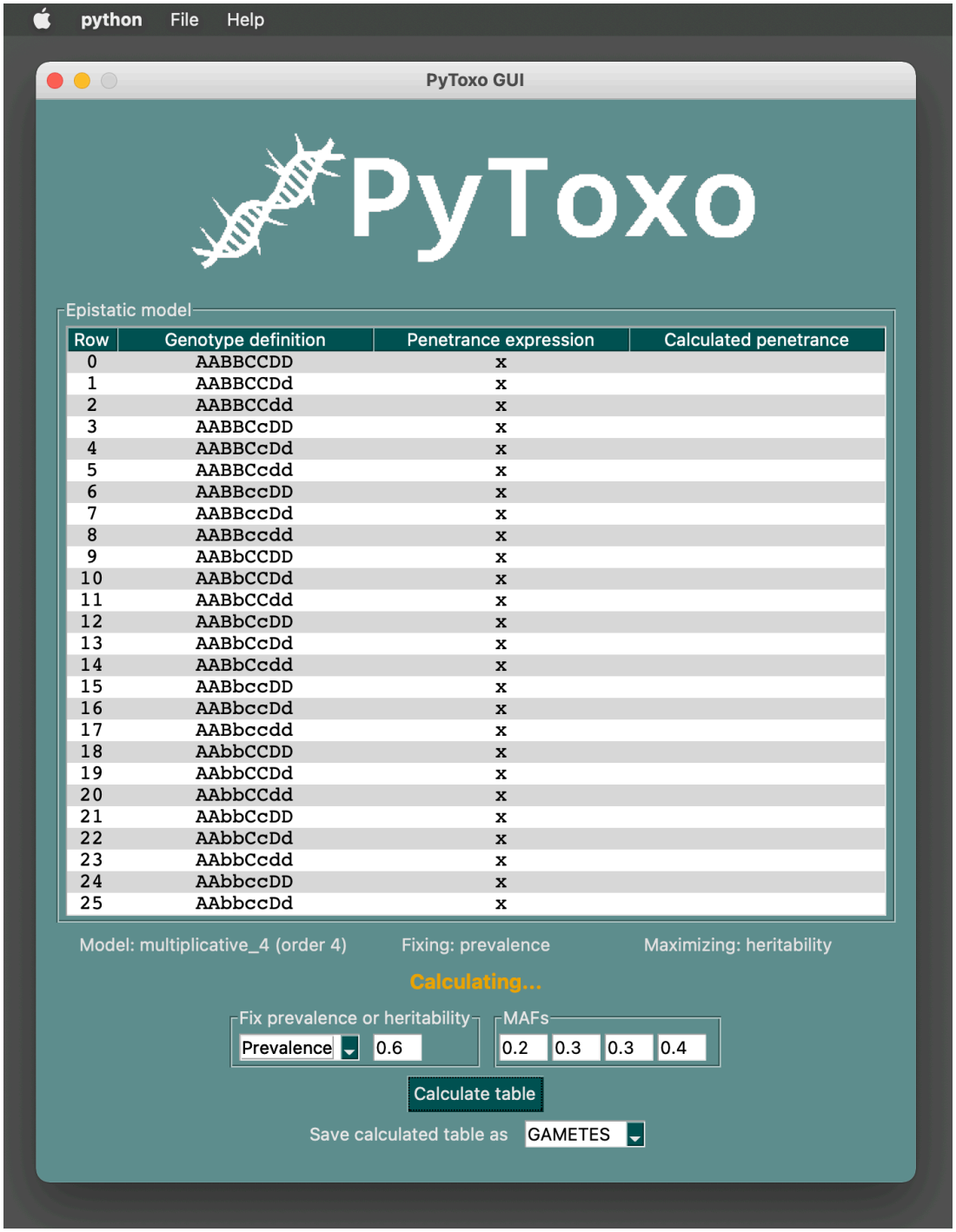

Figura 4.5: Pantalla de la GUI de PyToxo desde Mac OS, durante el proceso de cálculo de la tabla

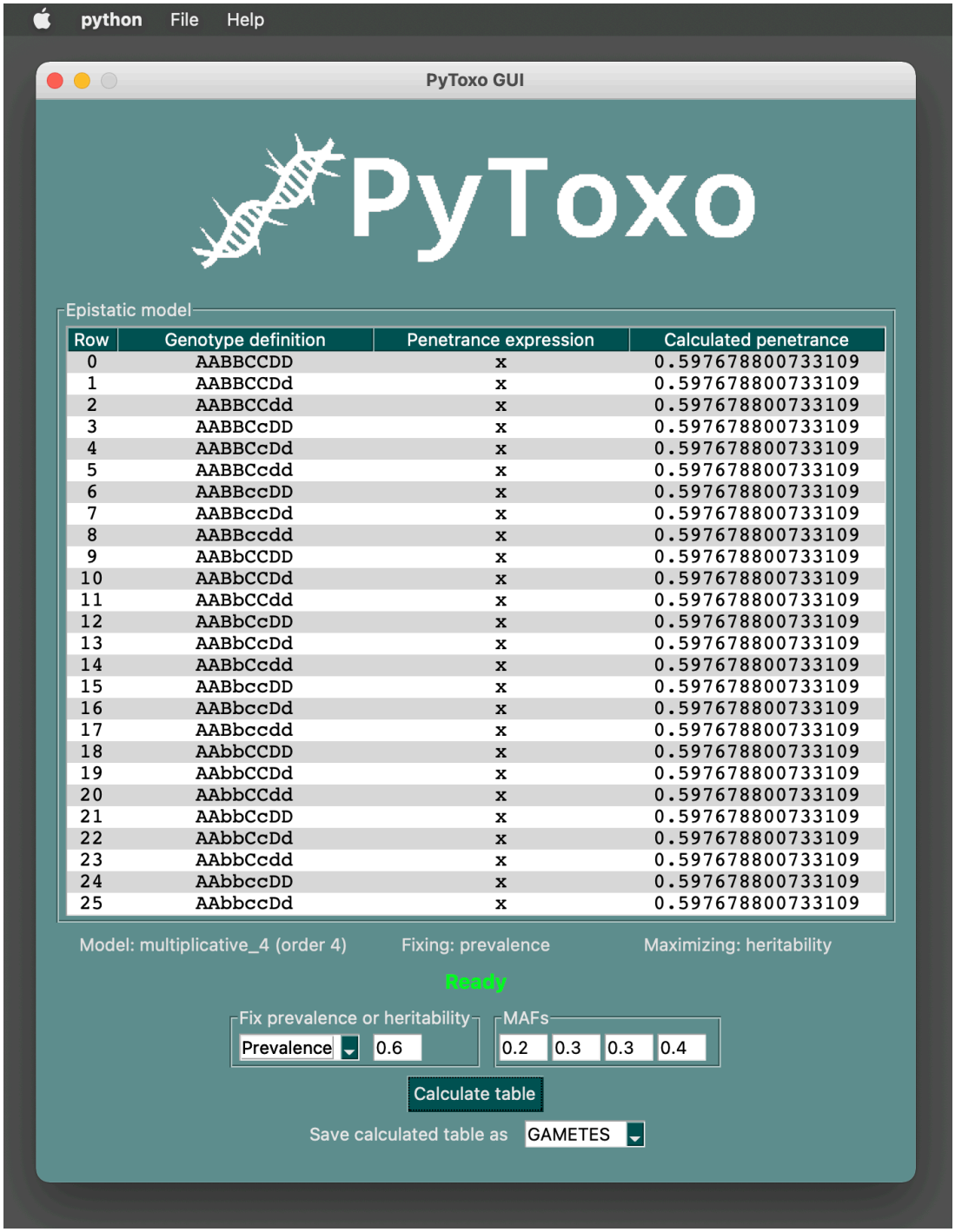

Figura 4.6: Pantalla final, con la tabla de penetrancia calculada, de la GUI de PyToxo desde Mac OS

Las figuras de la 4.2 a la 4.6 representan la versión de la interfaz para el sistema operativo Mac OS Big Sur, y en ellas queda patente la buena adaptación de la interfaz a las peculiaridades nativas de esta plataforma. Por ejemplo, apreciamos cómo los dos menús contextuales (*file* y *help*) se disponen en la barra superior, como es típico.

No obstante, como ya hemos mencionado, la interfaz gráfica de PyToxo es plenamente funcional también en Linux y Windows, integrándose en estos sistemas operativos como una aplicación nativa. En la Figura 4.7 presentamos la versión para Linux, en este caso desde una distribución Linux Lubuntu 20; y en la Figura 4.8 presentamos la propia de Windows, desde Windows 10. En ambos casos podemos apreciar las diferencias en la estética de la ventana y los botones, característicos de cada plataforma.

La GUI de PyToxo está desarrollada sobre Tk [44]. Tk es una librería de elementos básicos para construir una interfaz gráfica de usuario. Fue desarrollada como una extensión para el lenguaje Tcl, pero puede ser usada por otros lenguajes de programación. En Python, Tk se accede desde el *binding* Tkinter [32], y se considera un estándar para las interfaces gráficas en Python.

Tk ha sido portada para funcionar en la mayoría de las variantes de Linux, Apple Mac OS y Microsoft Windows. Tanto Tk como Tkinter están incluidas en la mayoría de las distribuciones modernas de Python para las anteriores plataformas. No ocurre así en algunas distribuciones de Linux, en las que hay que instalar un paquete adicional. Sin embargo, el propio PyToxo detecta esta carencia cuando se intenta lanzar la interfaz, y le sugiere al usuario el paquete a instalar para suplir la carencia y continuar trabajando inmediatamente.

De esta forma, la interfaz gráfica de PyToxo no sólo resulta altamente compatible, sino también muy distribuible, no llegando a necesitar ningún requerimiento fuera de Python que el usuario deba molestarse en satisfacer.

Aunque el motor de la interfaz gráfica de PyToxo sea Tk, como hemos expuesto, PyToxo no lo emplea directamente a través de Tkinter, sino que hace uso de una tercera librería para ello: PySimpleGUI [23]. El motivo de añadir esta dependencia es que con ella se simplifica notoriamente el desarrollo de la interfaz, pudiendo prescindir de ciertos ajustes propios de Tkinter más engorrosos e innecesarios para una pantalla sencilla como la de PyToxo. La implementación, contenida en el paquete `pytoxo_gui`, consta de un único módulo que no llega a las mil líneas de código.

Además, PySimpleGUI resulta una elección inteligente porque permitiría cambiar el motor subyacente de Tk a otro, como por ejemplo Qt [36], sin realizar apenas ajustes en el código propio, como de hecho manifiestan los propios desarrolladores de PySimpleGUI [23]. De esta forma, la interfaz también se ve reforzada en portabilidad.

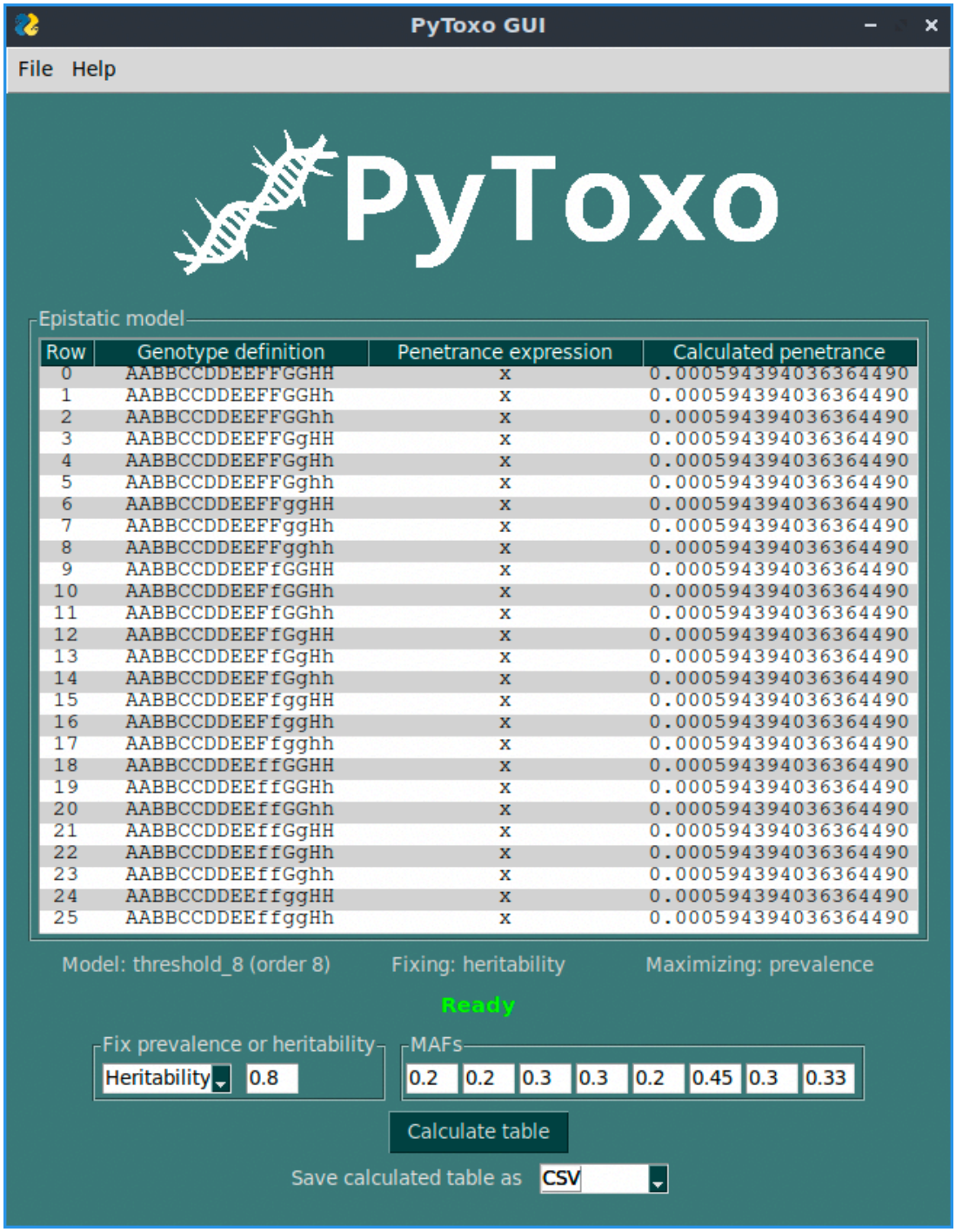

Figura 4.7: GUI de PyToxo desde Linux Lubuntu 20.04

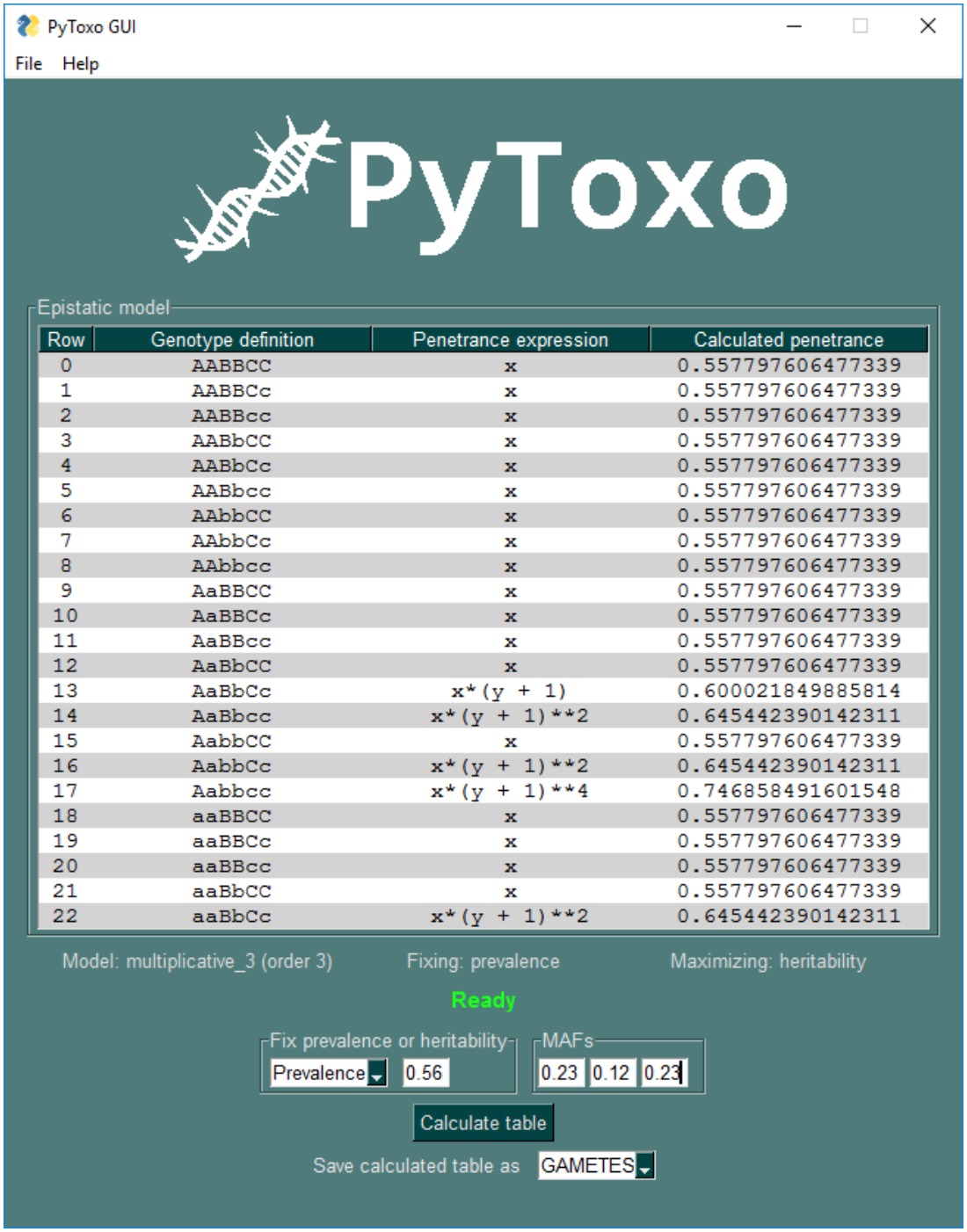

Figura 4.8: GUI de PyToxo desde Windows 10

Capítulo 5

Resultados y evaluación de PyToxo

En este capítulo se evaluará PyToxo en términos de alcance, precisión, tiempo de ejecución y usabilidad. Para ello, confrontaremos su implementación con la de Toxo.

Cabe señalar que, aunque PyToxo no está circunscrito de ninguna forma a ellos, para proceder con la evaluación contenida en este capítulo y para la implementación de las pruebas automáticas ya descritas en capítulos anteriores, se toman como referencia los ya mencionados y ampliamente utilizados modelos epistáticos de segundo orden de Marchini *et al.* [13] y sus generalizaciones de enésimo orden. Estos modelos están disponibles en el repositorio de PyToxo.

5.1 Alcance

En esta primera sección abordaremos el alcance de PyToxo, es decir, los modelos y configuraciones que es capaz de resolver. Como expusimos en la Sección 2.3 Toxo [21] sólo admite modelos que cumplan dos condiciones: ha de estar formado por expresiones monótonamente no decrecientes del espacio de los números reales positivos ($\mathbb{R}+$) y todas esas expresiones han de ser ordenables unívocamente en el mismo espacio $\mathbb{R}+$.

Sin embargo, estas dos restricciones son satisfechas por la mayor parte de los modelos de epistasia utilizados a día de hoy activamente en la bibliografía, como los modelos de Marchini *et al.* [13] o los modelos de heterogeneidad introducidos por Neuman *et al.* [15].

Como ya describimos en el Capítulo 3, PyToxo se basa en el método matemático de Toxo, por lo que hereda sus mismas limitaciones en relación a los modelos abordables. Así pues, la diferencia de alcance entre Pytoxo y Toxo vendrá dada por la capacidad de cada una de las herramientas de resolver los sistemas de ecuaciones que se conforman a partir de los modelos que cumplen las restricciones de entrada y los parámetros configurados.

En el prefacio del presente capítulo ya introdujimos que hemos utilizado los modelos de Marchini *et al.* [13] como pivote sobre el que basar la evaluación de PyToxo. Estos modelos

se basan en tres submodelos del caso general de dos *loci* bialélicos, y se parametrizan a partir del efecto base (α) —el efecto genético presente en cada locus independientemente de la combinación de alelos real— y el efecto genotípico (θ) —el aumento de las probabilidades de una determinada característica más allá del nivel del efecto base y debido a la interacción genética—. En la Tabla 5.1 detallamos cómo están formados los susodichos modelos, a partir de las tres variantes del de segundo orden.

El modelo de la Tabla 5.1a especifica que las probabilidades de la característica aumentan de forma multiplicativa tanto dentro como entre dos *loci*. En este modelo, un individuo que es heterocigoto en el locus A tiene mayores probabilidades de $1 + \theta$ en relación con las de un individuo que es homocigoto AA. Por su parte, el homocigoto aa tiene probabilidades adicionales, determinadas por $(1+\theta)^2$. Lo mismo ocurre para con el locus B si consideramos los mismos efectos sobre ambos *loci*, como es nuestro caso. Este es el denominado modelo aditivo.

El modelo de la Tabla 5.1b, nombrado como multiplicativo, es una interacción estadística que tiene efectos marginales explícitos en el que al menos un alelo asociado a la característica debe estar presente en cada locus para que las probabilidades vayan más allá del nivel del efecto base (α). A partir de ahí, cada copia adicional del alelo asociado a la característica en los *loci* A o B aumenta aún más las probabilidades por el factor multiplicativo $1 + \theta$.

Por último tenemos el modelo umbral, de la Tabla 5.1c. Este modelo adopta la misma forma que el multiplicativo de la Tabla 5.1b, al requerir al menos una copia de los alelos asociados a la característica en los *loci* A y B. Sin embargo, en este caso las copias adicionales de los alelos asociados a la característica no aumentan más el riesgo. Así, este modelo refleja los efectos del umbral de la característica, en los que se requiere una sola copia del alelo asociado con la misma en cada locus para incrementar las probabilidades de manifestarla, pero tener ambas copias del alelo asociado con la característica no tiene influencia adicional.

Para las pruebas de este capítulo partimos de los modelos originales de segundo orden y sus generalizaciones hasta el octavo orden de interacción. En la colección recopilada en el repositorio de PyToxo [1] recogemos los modelos de orden de 2 a 8 para cada una de esas tres variantes, como archivos CSV.

En cuanto al resto de parámetros para componer los casos; esto es, la MAF asociada a cada locus, la heredabilidad y la prevalencia; se ha intentado abarcar una muestra lo suficientemente representativa en las pruebas del abanico de valores posibles. Así, hemos trabajado con MAF comprendidas en el intervalo $[0, \frac{1}{2}]$, y heredabilidades y prevalencias en ambos casos dentro del intervalo $[0, 1]$.

Los estudios recientes que incluyen simulaciones basadas en modelos epistáticos para

[1] Modelos disponibles en `https://github.com/bglezseoane/pytoxo/tree/master/models` [6]

	BB	Bb	bb
AA	α	$\alpha(1+\theta)$	$\alpha(1+\theta)^2$
Aa	$\alpha(1+\theta)$	$\alpha(1+\theta)^2$	$\alpha(1+\theta)^3$
aa	$\alpha(1+\theta)^2$	$\alpha(1+\theta)^3$	$\alpha(1+\theta)^4$

(a) Modelo aditivo de orden 2

	BB	Bb	bb
AA	α	α	α
Aa	α	$\alpha(1+\theta)$	$\alpha(1+\theta)^2$
aa	α	$\alpha(1+\theta)^2$	$\alpha(1+\theta)^4$

(b) Modelo multiplicativo de orden 2

	BB	Bb	bb
AA	α	α	α
Aa	α	$\alpha(1+\theta)$	$\alpha(1+\theta)$
aa	α	$\alpha(1+\theta)$	$\alpha(1+\theta)$

(c) Modelo umbral de orden 2

Tabla 5.1: Modelo de Marchini *et al.* de orden 2, en sus tres variantes

generar sus datos de evaluación [1, 16, 52] se asientan en modelos de bajo orden y emplean valores de heredabilidad que tienden a ser también bajos. Esto se puede deber a la carencia de herramientas capaces de trabajar con modelos de alto orden y niveles paramétricos altos, como ya hemos comentado en el Capítulo 2. Aquí radica precisamente la innovación que Toxo [21] planteó en su día, y en este sentido PyToxo es plenamente continuista para con esta capacidad. PyToxo, junto con los simuladores actuales como GAMETES [50], puede facilitar los GWAS mediante la búsqueda de tablas de penetrancia adecuadas y la creación de muestras que se asemejan más a los datos del mundo real.

Tras observar el comportamiento de PyToxo con los diferentes bancos de pruebas, y a este respecto cabe destacar lo implementado en el subpaquete de pruebas `test.solubility`, podemos resumir el alcance de PyToxo como sigue:

- PyToxo es capaz de trabajar con modelos umbral de hasta orden 8, y probablemente con modelos umbral mayores.
- PyToxo es capaz de trabajar con modelos aditivos de hasta orden 8, y probablemente con modelos aditivos mayores.
- PyToxo es capaz de trabajar con modelos multiplicativos de hasta orden 5.

En los dos primeros casos de la enumeración anterior afirmamos que para los casos de modelos umbral y modelos aditivos es probable que PyToxo pueda acometer modelos de orden

mayor a 8, porque aunque no lo hemos llegado a probar, su desempeño con los modelos de hasta ese orden es muy ágil y todo parece indicar que todavía podría trabajar con casos de este tipo de mayor complejidad.

Los límites de la herramienta aparecen con los modelos de tipo multiplicativo, que por otra parte son los más complejos. Hasta el orden 5, PyToxo se desenvuelve bien, pero a partir de ese punto la mayoría de las configuraciones resultan irresolubles, debido a la excesiva acumulación de error en los procesos de cálculo internos de la librería, o bien porque se aborta el proceso al saberse que muy probablemente resulte en un error de no convergencia, como ya expusimos en la Sección 3.5.

No obstante, se han llegado a resolver algunas configuraciones concretas para el modelo multiplicativo de orden 6, como aprovechamos a incluir en la Tabla 5.3, lo cual nos sirve como ratificación a la hora de establecer esta frontera, que aunque difusa, existe.

Es importante hacer notar también que no todas las configuraciones de modelos, MAF y heredabilidad o prevalencia resultan resolubles. Hay ciertos casos en los que componen sistemas que son matemáticamente insatisfacibles, con lo que ni PyToxo ni Toxo encontrarán una solución.

A la hora de comparar el alcance de PyToxo con el de otras herramientas del estado del arte, como bien explicitamos a lo largo del Capítulo 2, Toxo es la principal referencia contra la que confrontarse. A este respecto, podemos afirmar que PyToxo es capaz de lidiar con más complejidad que Toxo. De hecho, el banco de pruebas del subpaquete `test.solubility` verifica que PyToxo es capaz de resolver al menos todo lo que Toxo, empleando una muestra obviamente no completa pero sí considerable de las configuraciones posibles. En concreto, se tienen en consideración 1474 casos, en los que en la mitad se maximiza la prevalencia y en la otra mitad la heredabilidad, y para ambos casos se representa todo el espectro posible para los valores de los parámetros.

Además, en las tablas 5.2 y 5.3 exponemos una muestra de casos particulares que PyToxo consigue resolver, mientras que Toxo concluye en tablas corruptas. Estas muestras han sido recortadas de informes generados por *scripts* del subpaquete de pruebas `test.solubility`, que en cada ejecución estudian un subconjunto aleatorio diferente de entre todas las configuraciones posibles.

Los datos para la elaboración de las tablas anteriores están disponibles para su consulta dentro del directorio `reports` del paquete `test.solubility` del repositorio de PyToxo. Al lector podría interesarle además consultar la carpeta `toxo_outputs` del repositorio, en la que consta una colección de tablas de penetrancia, registros de tiempos e informes de error que hemos recopilado al estudiar Toxo, y que empleamos para la elaboración de nuestros informes y finalmente, de estas tablas.

A la vista de los resultados, podemos concluir que PyToxo mejora significativamente el

Modelo	Orden	MAF	Heredabilidad
Aditivo	4	0,2	0,8
Aditivo	5	0,1	0,9
Aditivo	5	0,2	0,9
Aditivo	5	0,3	0,4
Aditivo	5	0,3	0,6
Aditivo	5	0,4	0,7
Aditivo	5	0,5	0,6
Aditivo	6	0,1	0,8
Aditivo	6	0,4	0,3
Aditivo	6	0,5	0,6
Aditivo	7	0,4	0,7
Aditivo	7	0,5	0,3
Aditivo	8	0,1	0,6
Aditivo	8	0,1	0,7
Aditivo	8	0,2	0,1
Aditivo	8	0,2	0,1
Aditivo	8	0,2	0,3
Aditivo	8	0,2	0,4
Aditivo	8	0,4	0,1
Aditivo	8	0,4	0,5

Tabla 5.2: Algunos casos que Toxo no puede resolver y PyToxo sí, maximizando la prevalencia

alcance proporcionado por Toxo.

5.2 Precisión

En la Sección 3.5 del Capítulo 3 avanzábamos que Pytoxo aplica una política de precisión en los resultados y que, a no ser que el usuario lo desactive, se comprueban siempre los cálculos para cerciorar que se cumple con la precisión establecida antes de componer una tabla de penetrancia. También hemos mencionado que PyToxo dispone de una batería de pruebas automáticas dedicadas en exclusiva a la precisión con la que se desempeña, contenida en el subpaquete de pruebas `test.accuracy`.

La precisión con la que PyToxo es capaz de trabajar depende en última instancia del orden del modelo que esté manejando. Cuanto mayor sea el modelo, mayor será el error acumulado, debido a que el sistema de ecuaciones a resolver se vuelve más complejo; y por tanto, menor será la precisión.

Para lidiar con esto, PyToxo utiliza una expresión matemática para calcular adaptativamente, en función del modelo, el delta de error, $d_{tolerable}$, que se considera tolerable para el caso que está tratando:

Modelo	Orden	MAF	Prevalencia
Multiplicativo	6	0,10	0,30
Multiplicativo	6	0,10	0,60
Multiplicativo	6	0,10	0,80
Multiplicativo	6	0,10	0,90
Multiplicativo	6	0,20	0,30
Multiplicativo	6	0,20	0,40
Multiplicativo	6	0,20	0,50
Multiplicativo	6	0,20	0,90
Multiplicativo	6	0,40	0,20
Multiplicativo	6	0,40	0,30

Tabla 5.3: Algunos casos que Toxo no puede resolver y PyToxo sí, maximizando la heredabilidad

$$d_{tolerable} = \min\left(10^{orden} d_0, d_{max}\right) \tag{5.1}$$

Esta expresión la hemos obtenido heurísticamente al observar cómo se comporta el programa. En ella distinguimos dos parámetros que están fijados como constantes en la implementación:

- d_0: el delta base, fijado a 1×10^{-16}.
- d_{max}: el delta máximo, fijado a 1×10^{-8}.

El delta de error (d_{error}) del caso en concreto lo hallamos tomando las soluciones de las dos variables que ha calculado el programa y substituyéndolas de nuevo en sus sistemas de origen, esto es, en la Ecuación 2.4 o la Ecuación 2.6. Los segundos miembros de las ecuaciones de estos sistemas son constantes. Así pues, hallamos la desviación entre el segundo miembro que resulta al substituir en el primer miembro el nuevo valor hallado y el segundo miembro real que conocemos. El delta final a considerar como d_{error} es el máximo de los dos resultantes. O dicho de otra forma, el d_{error} será la máxima diferencia entre los valores de heredabilidad o prevalencia originalmente establecidos y los obtenidos al completar la substitución.

Si d_{error} resultase mayor que $d_{tolerable}$, se consideraría que la solución no es válida y se generaría un error de resolución para advertir al usuario.

Así, las soluciones que halle PyToxo tendrán garantizada como máximo una desviación tolerable del orden de magnitud 10^{-8}, para modelos de orden 8 o superior; y como mínimo una de 10^{-14}, para modelos de orden 2.

No obstante lo anterior, estas desviaciones son las que están garantizadas, pero PyToxo, por norma general, consigue unos resultados mejores, que rara vez presentan desviaciones superiores al orden de magnitud de 10^{-16}.

En la Tabla 5.4 recogemos una comparativa a nivel de precisión entre PyToxo y Toxo. En ella se estudian modelos desde el segundo hasta el octavo orden, de los tres tipos posibles. Para cada uno de esos modelos se han empleado MAF de 0,1 y 0,4, y para cada una de las combinaciones de modelos y MAF, heredabilidades de 0,1 y 0,8. Así, se alcanza una muestra bastante amplia de casos a estudiar.

Las configuraciones que no aparecen en la tabla son aquellas para las que o bien PyToxo o bien Toxo no han alcanzado una solución válida. En términos generales, PyToxo, como ya hemos expuesto en la sección anterior, comienza a experimentar problemas para obtener soluciones válidas cuando utilizamos modelos multiplicativos de orden 5 o mayores. Para modelos menos complejos, como los aditivos, en los que es fácil observar bastantes ausencias en la tabla, la falta de casos se debe a errores en Toxo.

PyToxo consigue una precisión mejor o igual que Toxo en 39 de los 50 casos mostrados en la Tabla 5.4. Además, el error medio cometido por PyToxo resulta en $5{,}75 \times 10^{-17}$ y el de Toxo en $1{,}78 \times 10^{-4}$, con lo que podemos afirmar que PyToxo mejora notoriamente la precisión de las soluciones que alcanza con respecto a su predecesor Toxo.

Los datos recogidos para la elaboración de la tabla anterior están almacenados y disponibles para su consulta dentro del subdirectorio `reports` del paquete `test.accuracy` del repositorio de PyToxo. Al lector también podría interesarle consultar la carpeta `toxo_outputs`, en la que hemos recolectado una colección de tablas de penetrancia, registros de tiempos de ejecución e informes de error de Toxo.

5.3 Tiempo de ejecución

Al que igual con la precisión, el tiempo que requiere PyToxo para calcular una tabla de penetrancia crece cuanto mayor sea el orden del modelo a utilizar. Sin embargo, para los modelos que quedan dentro del alcance de la herramienta, PyToxo se desenvuelve muy ágilmente, tardando en todos los casos menos de un minuto. También en este aspecto, PyToxo mejora a Toxo, como presentamos en la Tabla 5.5. En ese conjunto de casos, bastante representativo, se ha logrado una razón de aceleración media de 1,90 de PyToxo con respecto a Toxo, lográndose una aceleración por encima de 10 para los modelos más complejos (modelos multiplicativos de orden 4).

Cabe señalar que la información recogida en la Tabla 5.5 se ha recopilado corrigiendo todos los tiempos, calculando la media aritmética de varias ejecuciones para descartar interferencias debidas a otros procesos existentes en la máquina en ese momento. Tanto los casos de PyToxo como los de Toxo se han ejecutado en la misma máquina, con procesador Intel Core i5-9600K de seis núcleos a 3,7 GHz y 16 GB de memoria RAM DDR4. La versión de MATLAB [46] empleada para ejecutar Toxo ha sido la R2020b. El sistema operativo instalado en la máquina

era Apple Mac OS Big Sur 11.2.1.

En la Tabla 5.4 se muestran para las mismas configuraciones que en la sección anterior los tiempos de ejecución utilizando PyToxo y Toxo.

5.4 Usabilidad

La usabilidad es sin duda la característica más subjetiva de entre las que consideramos en el presente capítulo a la hora de evaluar PyToxo. Sin embargo, creemos que queda patente nuestro esfuerzo para intentar que esta herramienta pueda ser empleada por usuarios de diversos ámbitos fácilmente, con una curva de aprendizaje muy poco pronunciada.

Hemos implementado una librería muy sencilla de utilizar y con un código fuente profusamente comentado siguiendo las recomendaciones oficiales de la Python Software Foundation [27]. Por todo el código de PyToxo nos encontramos las anotaciones de tipos introducidas en Python 3.5 [33] para explicitar el tipo de los diferentes parámetros implicados, así como cabeceras muy detalladas (*docstrings*, en jerga Python) para todas las funciones y métodos, como exponemos en el ejemplo del Código 5.1 [2], siguiendo la convención de estilo de NumPy [47]. Además, todo el código fuente está formateado por la herramienta Black [53], siguiendo las convenciones oficiales recogidas en la PEP 8 de Python [28].

Por todo lo anterior, consideramos que PyToxo es una librería bien adaptada para su uso y explotación por parte de usuarios programadores, que podrían comprender toda la funcionalidad navegando por un código del que se ha cuidado especialmente su legibilidad, como acabamos de defender.

En cuanto a las interfaces, como demostramos en la Sección 3.6, la interfaz en línea de comando está muy adaptada a lo que es habitual en este tipo de herramientas en entornos POSIX [8], por lo que un usuario con experiencia en este tipo de utilidades la encontrará muy natural.

Puede que lo más destacable del apartado de la usabilidad sea la implementación de una interfaz gráfica que hace que PyToxo pueda ser empleado por usuarios sin conocimientos de programación, ni demasiada cercanía con entornos en línea de comandos. La GUI acometida resulta muy fácil de utilizar gracias a su disposición minimalista y a que va guiando al usuario en el proceso, restringiendo aquello que puede hacer.

Como el lector ha podido apreciar en los ejemplos incluidos a lo largo de este documento, tanto la CLI como la GUI están en inglés, para intentar llegar al máximo número de usuarios posibles. Como característica que mejorar en materia de usabilidad se podría considerar la traducción a más idiomas de ambas interfaces. Lo que sí está disponible al menos tanto

[2] Método encargado de calcular el delta de error tolerable que tratamos en la Sección 5.2, por medio de la Ecuación 5.1

```
def calculate_tolerable_solution_error_delta(self) -> float:
    """Calculates the error delta to be tolerated during the model
    resolution, adjusted to the current model order. If the adjusted
    delta is greater than the maximum delta, returns the maximum
    delta. The latter avoid absurd delta error for very large
    models.

    Uses the constants `_TOLERABLE_SOLUTION_ERROR_BASE_DELTA` and
    `_TOLERABLE_SOLUTION_ERROR_MAX_DELTA` to calculate the fitted
    delta.

    Returns
    -------
    float
        The resolution error delta to tolerate.
    """
```

Código 5.1: Cabecera de `Model.calculate_tolerable_solution_error_delta`

en castellano como en inglés es el manual de usuario, que en el presente documento hemos incluido respectivamente en los anexos A y B.

Por último, también queremos hacer notar que PyToxo puede instalarse en cualquier máquina que cuente con una distribución normal de Python 3.8 o superior haciendo uso de un solo comando, lo cual facilita considerablemente su empleo a todo tipo de usuarios.

Así pues, considerando todo lo expuesto, mantenemos nuestra postura inicial y manifestamos que PyToxo es una herramienta bien elaborada en cuanto a usabilidad se refiere. Creemos también que mejora a Toxo en materia de usabilidad, al contar con interfaces de usuario que resultan mucho más accesibles que el empleo directo del código fuente.

Modelo	Orden	MAF	Heredabilidad	Error PyToxo	Error Toxo	Mejor precisión
Umbral	2	0,1	0,1	$2{,}78 \times 10^{-17}$	$2{,}45 \times 10^{-8}$	PyToxo
Umbral	2	0,1	0,8	0	0	Empate
Umbral	2	0,4	0,1	$1{,}11 \times 10^{-16}$	0	Toxo
Umbral	2	0,4	0,8	$1{,}11 \times 10^{-16}$	0	Toxo
Umbral	3	0,1	0,1	$1{,}39 \times 10^{-17}$	$1{,}70 \times 10^{-7}$	PyToxo
Umbral	3	0,1	0,8	0	$3{,}44 \times 10^{-9}$	PyToxo
Umbral	3	0,4	0,1	0	0	Empate
Umbral	3	0,4	0,8	$1{,}11 \times 10^{-16}$	0	Toxo
Umbral	4	0,1	0,1	0	$7{,}41 \times 10^{-7}$	PyToxo
Umbral	4	0,1	0,8	0	$1{,}60 \times 10^{-8}$	PyToxo
Umbral	4	0,4	0,1	$2{,}78 \times 10^{-17}$	0	Toxo
Umbral	4	0,4	0,8	0	0	Empate
Umbral	5	0,1	0,1	0	$1{,}41 \times 10^{-6}$	PyToxo
Umbral	5	0,1	0,8	$1{,}11 \times 10^{-16}$	$5{,}71 \times 10^{-8}$	PyToxo
Umbral	5	0,4	0,1	$1{,}39 \times 10^{-17}$	0	Toxo
Umbral	5	0,4	0,8	$1{,}11 \times 10^{-16}$	$4{,}17 \times 10^{-9}$	PyToxo
Umbral	6	0,1	0,1	$1{,}39 \times 10^{-17}$	$1{,}95 \times 10^{-5}$	PyToxo
Umbral	6	0,1	0,8	$2{,}22 \times 10^{-16}$	$5{,}43 \times 10^{-8}$	PyToxo
Umbral	6	0,4	0,1	$1{,}39 \times 10^{-17}$	0	Toxo
Umbral	6	0,4	0,8	0	$5{,}05 \times 10^{-9}$	PyToxo
Umbral	7	0,1	0,1	$1{,}39 \times 10^{-17}$	$3{,}81 \times 10^{-5}$	PyToxo
Umbral	7	0,1	0,8	$1{,}11 \times 10^{-16}$	$1{,}58 \times 10^{-8}$	PyToxo
Umbral	7	0,4	0,1	$4{,}16 \times 10^{-17}$	0	Toxo
Umbral	7	0,4	0,8	$2{,}22 \times 10^{-16}$	0	Toxo
Umbral	8	0,1	0,1	$2{,}78 \times 10^{-17}$	$2{,}42 \times 10^{-4}$	PyToxo
Umbral	8	0,1	0,8	$1{,}11 \times 10^{-16}$	$1{,}29 \times 10^{-6}$	PyToxo
Umbral	8	0,4	0,1	$1{,}39 \times 10^{-17}$	$3{,}51 \times 10^{-8}$	PyToxo
Umbral	8	0,4	0,8	$1{,}11 \times 10^{-16}$	0	Toxo
Aditivo	3	0,1	0,1	$1{,}39 \times 10^{-17}$	$1{,}05 \times 10^{-6}$	PyToxo
Aditivo	3	0,1	0,8	$2{,}22 \times 10^{-16}$	$3{,}26 \times 10^{-5}$	PyToxo
Aditivo	3	0,4	0,1	$9{,}71 \times 10^{-17}$	0	Toxo
Aditivo	3	0,4	0,8	$1{,}11 \times 10^{-16}$	$1{,}62 \times 10^{-7}$	PyToxo
Aditivo	4	0,1	0,1	$2{,}78 \times 10^{-17}$	$5{,}54 \times 10^{-4}$	PyToxo
Aditivo	4	0,4	0,1	$1{,}39 \times 10^{-17}$	0	Toxo
Aditivo	4	0,4	0,8	0	$3{,}92 \times 10^{-3}$	PyToxo
Aditivo	5	0,4	0,1	$1{,}39 \times 10^{-17}$	$2{,}66 \times 10^{-8}$	PyToxo
Aditivo	6	0,4	0,1	$1{,}39 \times 10^{-17}$	$6{,}37 \times 10^{-7}$	PyToxo
Aditivo	7	0,4	0,1	0	$4{,}69 \times 10^{-7}$	PyToxo
Multiplicativo	2	0,1	0,1	0	$4{,}17 \times 10^{-6}$	PyToxo
Multiplicativo	2	0,1	0,8	0	$1{,}20 \times 10^{-7}$	PyToxo
Multiplicativo	2	0,4	0,1	0	0	Empate
Multiplicativo	2	0,4	0,8	0	$4{,}88 \times 10^{-9}$	PyToxo
Multiplicativo	3	0,1	0,1	$1{,}25 \times 10^{-16}$	$7{,}51 \times 10^{-4}$	PyToxo
Multiplicativo	3	0,1	0,8	0	$3{,}59 \times 10^{-6}$	PyToxo
Multiplicativo	3	0,4	0,1	$4{,}16 \times 10^{-17}$	$1{,}63 \times 10^{-7}$	PyToxo
Multiplicativo	3	0,4	0,8	$1{,}11 \times 10^{-16}$	$1{,}39 \times 10^{-8}$	PyToxo
Multiplicativo	4	0,1	0,1	$8{,}33 \times 10^{-17}$	$3{,}29 \times 10^{-3}$	PyToxo
Multiplicativo	4	0,1	0,8	$3{,}33 \times 10^{-16}$	$3{,}57 \times 10^{-5}$	PyToxo
Multiplicativo	4	0,4	0,1	$1{,}39 \times 10^{-17}$	$1{,}08 \times 10^{-6}$	PyToxo
Multiplicativo	4	0,4	0,8	$1{,}11 \times 10^{-16}$	$3{,}60 \times 10^{-8}$	PyToxo

Tabla 5.4: Comparativa de precisión entre PyToxo y Toxo con casos resolubles por ambos

Modelo	Orden	MAF	Heredabilidad	Tiempo PyToxo	Tiempo Toxo	Aceleración
Umbral	2	0,1	0,1	0,20	0,55	2,70
Umbral	2	0,1	0,8	0,18	0,23	1,22
Umbral	2	0,4	0,1	0,14	0,24	1,70
Umbral	2	0,4	0,8	0,14	0,22	1,57
Umbral	3	0,1	0,1	0,39	0,31	0,79
Umbral	3	0,1	0,8	0,37	0,71	1,91
Umbral	3	0,4	0,1	0,31	0,30	0,98
Umbral	3	0,4	0,8	0,32	0,29	0,92
Umbral	4	0,1	0,1	0,81	0,52	0,65
Umbral	4	0,1	0,8	0,80	0,92	1,15
Umbral	4	0,4	0,1	0,67	0,52	0,77
Umbral	4	0,4	0,8	0,67	0,53	0,79
Umbral	5	0,1	0,1	1,95	1,99	1,02
Umbral	5	0,1	0,8	2,04	2,34	1,15
Umbral	5	0,4	0,1	2,39	1,98	0,83
Umbral	5	0,4	0,8	2,43	1,94	0,80
Umbral	6	0,1	0,1	4,70	4,31	0,92
Umbral	6	0,1	0,8	4,68	4,26	0,91
Umbral	6	0,4	0,1	6,64	4,29	0,65
Umbral	6	0,4	0,8	6,59	4,70	0,71
Umbral	7	0,1	0,1	11,69	18,27	1,56
Umbral	7	0,1	0,8	11,62	18,68	1,61
Umbral	7	0,4	0,1	18,50	18,33	0,99
Umbral	7	0,4	0,8	18,49	18,70	1,01
Umbral	8	0,1	0,1	28,72	39,03	1,36
Umbral	8	0,1	0,8	28,70	39,23	1,37
Umbral	8	0,4	0,1	52,58	38,91	0,74
Umbral	8	0,4	0,8	52,69	38,88	0,74
Aditivo	3	0,1	0,1	2,13	3,04	1,43
Aditivo	3	0,1	0,8	1,90	1,67	0,88
Aditivo	3	0,4	0,1	1,81	1,53	0,85
Aditivo	3	0,4	0,8	1,76	1,62	0,92
Aditivo	4	0,1	0,1	3,04	8,72	2,87
Aditivo	4	0,4	0,1	3,16	4,94	1,56
Aditivo	4	0,4	0,8	3,20	4,72	1,47
Aditivo	5	0,4	0,1	6,14	8,80	1,43
Aditivo	6	0,4	0,1	13,19	22,60	1,71
Aditivo	7	0,4	0,1	25,86	51,82	2,00
Multiplicativo	2	0,1	0,1	0,67	2,84	4,25
Multiplicativo	2	0,1	0,8	0,69	1,01	1,47
Multiplicativo	2	0,4	0,1	0,71	1,01	1,43
Multiplicativo	2	0,4	0,8	0,64	1,00	1,56
Multiplicativo	3	0,1	0,1	1,62	5,88	3,62
Multiplicativo	3	0,1	0,8	1,62	3,95	2,44
Multiplicativo	3	0,4	0,1	1,66	2,96	1,78
Multiplicativo	3	0,4	0,8	1,59	3,48	2,18
Multiplicativo	4	0,1	0,1	4,69	83,18	17,75
Multiplicativo	4	0,1	0,8	4,34	81,94	18,88
Multiplicativo	4	0,4	0,1	4,56	54,42	11,94
Multiplicativo	4	0,4	0,8	4,77	51,33	10,75

Tabla 5.5: Comparativa de tiempo de ejecución entre PyToxo y Toxo con casos resolubles por ambos. Los tiempos están en segundos. La acelaración ha sido calculada diviendo el tiempo de Toxo por el de PyToxo

Capítulo 6

Planificación y costes

En este capítulo se presenta minuciosamente la planificación de este Trabajo de Fin de Grado. Se parte de una descripción de las fases y tareas en las que se dividió el proyecto, pasando después a mostrar la cronología de las mismas. El capítulo se cierra con una estimación de los costes derivados del proyecto.

6.1 Fases del proyecto

Para la realización de este Trabajo de Fin de Grado se ha empleado un modelo de desarrollo clásico en cascada. Los motivos que han propiciado esta elección son que se conocen de antemano y de forma clara las funcionalidades que debe tener el producto final, y que la base del producto final no es divisible en partes que pudieran constituir ningún producto mínimo viable que ir extendiendo —PyToxo debe, esencialmente, reemplazar la funcionalidad de Toxo y mejorar sus prestaciones—.

No obstante lo anterior, para algunas tareas se ha flexibilizado la secuencialidad para permitir solapamientos que beneficien el desarrollo del proceso. Por ejemplo, la implementación de las baterías de test de verificación y rendimiento se solapan parcialmente con la traducción de la herramienta.

A continuación se presenta el plan detallado atendiendo a este enfoque en cascada, describiendo las fases en las que se ha dividido la realización de este Trabajo de Fin de Grado.

6.1.1 Análisis del ámbito y del *software* original (F1)

La fase de análisis abarca lo relativo al estudio de bibliografía y documentación sobre el ámbito del proyecto, el estado del arte del propósito final del producto y el código de la versión original en MATLAB de la herramienta.

Una vez concluida esta fase, se habrá asentado el contexto en el que se desarrolla este Trabajo de Fin de Grado y se habrá llegado a una especificación clara del producto final PyToxo.

6.1.2 Traducción de la herramienta de MATLAB a Python (F2)

Una vez concretados los requisitos del producto *software* final y explorado el ámbito del mismo, es momento de traducir la herramienta original Toxo, escrita en MATLAB, a su versión PyToxo, en Python. A la vista de las decisiones que en su día se tomaron durante el diseño de Toxo, se trazará un diseño propio que tenga en cuenta las peculiaridades de Python y los objetivos de extensión con los que PyToxo contará con respecto a Toxo. Durante este proceso será necesario explorar las diferentes opciones de librerías disponibles en Python para suplir las empleadas en la versión original MATLAB. Acto seguido se implementará el código en Python.

Como parte de esta fase también se construirá una batería de test para cerciorar el correcto funcionamiento de PyToxo, usando como referencia para los resultados finales la comparativa con Toxo ante un banco de ejemplos dado.

6.1.3 Revisión de pruebas de rendimiento (F3)

La fase anterior abarca pruebas para comprobar el correcto funcionamiento de la herramienta. Esta, en cambio, se centra en desarrollar otra batería de test para establecer una comparativa de rendimiento entre PyToxo y Toxo.

De ser detectado un desvío muy importante que comprometa la eficiencia del producto, se podrían explorar cambios en el *software* para tratar de paliar ese defecto (por ejemplo, substitución de librerías internas u optimización de cálculos). No obstante, esta fase se ha separado de la anterior porque, aunque haya que realizar cambios internos, el *software* ya no debería de sufrir más variaciones superficiales, con lo que sería posible paralelizar esta fase con otras.

6.1.4 Adición de componentes de usabilidad (F4)

Uno de los objetivos de mejora de PyToxo con respecto a Toxo es facilitar su usabilidad. Así, dentro de esta fase se emprenderán tareas relativas a presentar una librería Python —enfocada al uso de PyToxo por otras utilidades—, una interfaz por línea de comando potente y versátil —enfocada a usuarios avanzados— y una interfaz gráfica —enfocada a usuarios menos familiarizados con la línea de comando y para mejorar la visualización de la salida del programa—.

6.1.5 Documentación y redacción de la memoria (F5)

Durante las fases anteriores se habrán ido recopilando notas y generando alguna documentación básica. Esta última fase del proyecto consiste en reunir todos esos apuntes para

componer un manual de usuario para PyToxo, además de para concluir con la redacción del texto del presente Trabajo de Fin de Grado.

6.2 Tareas desglosadas

En la Tabla 6.1 se presenta un desglose de las tareas identificadas, así como una estimación de las horas que se dedicarán a cada una de ellas. Para este Trabajo de Fin de Grado se han estimado 300 horas para su realización —a cargo del estudiante— y 50 horas para la supervisión del mismo —a cargo de sus directores—.

Fase	Tarea	Descripción	Horas
	T0	Planificación del proyecto	5
F1	T1	Búsqueda y estudio de bibliografía	15
F1	T2	Análisis del *software* original Toxo	20
F2	T3	Traducción de la herramienta MATLAB a Python	90
F2	T4	Creación de la batería de test de verificación	15
F3	T5	Creación de la batería de test de rendimiento	10
F4	T6	Implementación de la interfaz de librería Python documentada	10
F4	T7	Implementación de la interfaz en línea de comando	10
F4	T8	Implementación de la interfaz gráfica	20
F5	T9	Composición del manual de usuario	5
F5	T10	Composición de la memoria	100
	TS	Supervisión del trabajo durante todo su desarrollo	50
		Total	350

Tabla 6.1: Desglose inicial de las tareas del proyecto

6.2.1 Desviaciones de la planificación

Como en todo proyecto, en el desarrollo de este Trabajo de Fin de Grado también se han producido ciertos desajustes entre la planificación inicial y el transcurso final del mismo. En la Tabla 6.2 se presenta un desglose de las desviaciones en horas de cada una de las tareas.

Tarea	Horas estimadas	Horas finales	Desviación (horas)
T0	5	5	0
T1	15	15	0
T2	20	25	5
T3	90	100	10
T4	15	15	0
T5	10	10	0
T6	10	10	0
T7	10	10	0
T8	20	30	10
T9	5	5	0
T10	100	125	25
TS	50	50	0
Total	350	400	50

Tabla 6.2: Desviaciones en la planificación de las tareas del proyecto

Tal y como apreciamos en la tabla anterior, finalmente han sido necesarias 50 horas adicionales para completar el proyecto, de forma que han sido invertidas 350 horas para su realización —a cargo del estudiante— y 50 horas para la supervisión del mismo —a cargo de sus directores—, conllevando un total de 400 horas. Consideramos pues, que la planificación inicial estaba bastante bien ajustada.

6.2.2 Cronograma actualizado

El proyecto se inicia oficialmente el día 3 de diciembre de 2020, y en un primer momento el objetivo es el de presentar este Trabajo de Fin de Grado a la convocatoria de defensa de junio de 2021.

Lo anterior implica un total de 28 semanas para el desarrollo de este Trabajo de Fin de Grado, lo cual supone unas 16 horas de trabajo semanales de media durante la realización del mismo. El solapamiento de la línea temporal del trabajo con períodos de vacaciones y de exámenes a los que el estudiante ha de concurrir, hace que el esfuerzo se distribuya de la forma que mejor convenga en cada momento dado y no homogéneamente, con lo que la carga de horas por semana es variable.

En la Figura 6.1 se presenta un cronograma detallado en el que se visualiza la planificación de la ejecución de las tareas desglosadas en la Tabla 6.1, además de las dependencias entre las tareas. La Tabla 6.3 complementa esta información. Ambas tablas contienen información actualizada de lo que finalmente se ha llevado a cabo en la realización de este Trabajo de Fin

Tarea	Fecha de inicio	Fecha de fin
T1	7/12/2020	19/12/2020
T2	20/12/2020	15/1/2021
T3	18/1/2021	1/3/2021
T4	19/2/2021	10/3/2021
T5	5/3/2021	19/3/2021
T6	22/3/2021	26/3/2021
T7	5/4/2021	9/4/2021
T8	12/4/2021	23/4/2021
T9	19/4/2021	29/4/2021
T10	22/3/2021	22/6/2021

Tabla 6.3: Detalle de las fechas de cada tarea en la planificación

de Grado.

Sobre las relaciones de precedencia de las tareas, cabe comentar lo siguiente:

- T1 y T2 se realizan de forma secuencial, con dependencia fin-comienzo.
- Una vez terminada T2, se puede iniciar T3.
- T3 está relacionada comienzo-comienzo y fin-fin con T4. T4 está relacionada comienzo-comienzo y fin-fin con T5. Estas tres tareas se realizan en cascada pero con cierto solapamiento para permitir que se enriquezcan entre sí en el proceso de desarrollo. Es más fácil escribir las pruebas para un *software* justo después de su implementación que habiendo transcurrido un tiempo mayor. Así, se avanza T3 durante un par de semanas, y antes de que concluya ya se va trabajando en T4. Lo mismo ocurre con T5, que se inicia cuando T4 está avanzada pero no finalizada.
- Con T4 se alcanza un *software* estable, lo que permite iniciar las tareas relativas a la implementación de las interfaces programática y de usuario (tanto en línea de comando como gráfica). Así, la finalización de esta tarea habilita el inicio de T6, T7 y T8, respectivamente. También permite ir adelantando la composición de la documentación, con lo que se da paso a T9.
- T10 se inicia con suficiente holgura como para poder terminarla a tiempo y no puede terminar hasta que concluyan el resto de tareas del proyecto, con lo que se establecen relaciones fin-fin con T4, T5, T6, T7, T8 y T9.
- TS, la tarea de supervisión, se mantiene durante todo el transcurso del proyecto.

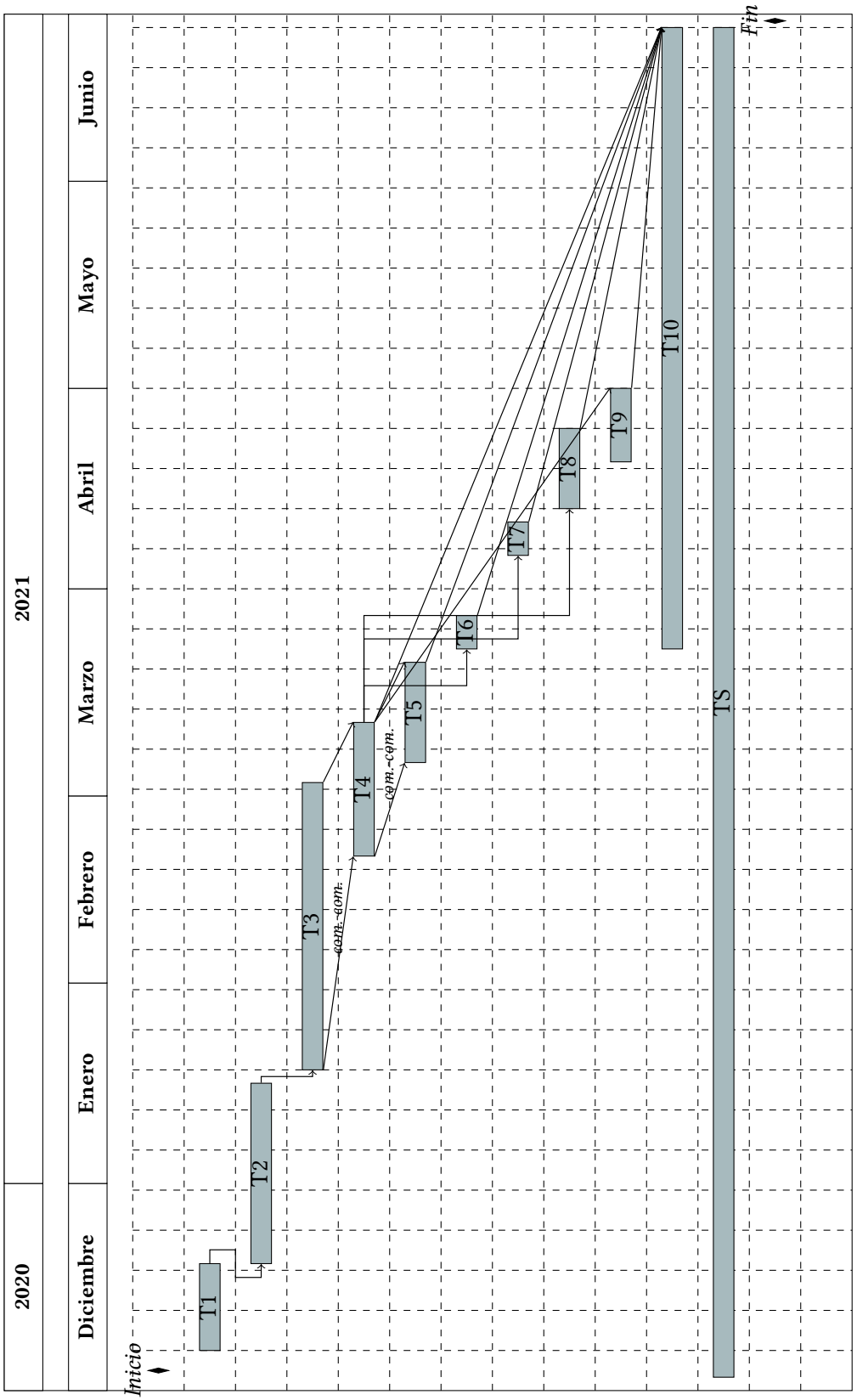

Figura 6.1: Diagrama de Gantt con el cronograma de las tareas del proyecto

6.3 Presupuesto

En el desarrollo de este Trabajo de Fin de Grado participaron tres personas: el estudiante, encargado de la realización del proyecto, y los dos directores, encargados de la supervisión del mismo. En la Tabla 6.4 se presenta una estimación de las horas a dedicar al proyecto de cada recurso, su coste por hora y los sumatorios que permiten obtener el coste total aproximado que habría tenido este trabajo.

Recurso	**Horas estimadas**	**Coste por hora (€/h)**	**Coste (€)**
Estudiante	350	40	14 000
Director 1	25	60	1500
Director 2	25	60	1500
	Total		17 000

Tabla 6.4: Costes del proyecto

Capítulo 7

Conclusiones

En este capítulo se presentan las conclusiones alcanzadas con este Trabajo de Fin de Grado, su relación con las competencias de la titulación de la que es colofón y las posibles líneas de trabajo futuras que se podrían abrir a raíz el mismo.

7.1 Conclusiones

El objetivo principal de este Trabajo de Fin de Grado no era otro que la reimplementación de la librería Toxo en el lenguaje de programación Python, alcanzando una nueva aplicación denominada PyToxo que mejorara sus carencias a la vez que se basaba en su aproximación matemática, innovadora en el estado del arte. Tras haber concluido el proyecto, y con él el desarrollo de PyToxo, afirmamos que los objetivos se han cumplido satisfactoriamente.

Creemos haber alcanzado una herramienta útil para los profesionales que se dedican a la investigación en el ámbito de los GWAS. PyToxo es un *software* estable y que funciona de hasta tres formas diferentes: como librería, mediante interfaz en línea de comando y mediante interfaz gráfica. Esta última lo acerca especialmente a aquellos usuarios menos instruidos en la programación o en entornos de ejecución en línea de comandos, permitiendo una curva de aprendizaje muy poco pronunciada.

Aunque la usabilidad del producto final era una faceta importante del proyecto, como bien hemos expuesto; PyToxo no sólo mejora a Toxo en esa característica. PyToxo incrementa muy notablemente el alcance de Toxo y el resto de herramientas similares del estado del arte. Es capaz de trabajar con modelos de mayor orden, así como más combinaciones de modelos y parámetros.

También es un *software* más preciso que su antecesor. PyToxo comete un error menor y mucho más regular que Toxo en la inmensa mayoría de las pruebas que se han llevado a cabo. Su error medio se sitúa en el orden de magnitud 10^{-17}, mientras que el error máximo que garantiza para aceptar una solución como válida nunca sobrepasará el orden 10^{-8} —y eso

sólo para los modelos más complejos—. El error medio de Toxo es mucho más oscilante que el de PyToxo, y por poner una cifra, en las pruebas presentadas en la Sección 5.2, resultaba en el orden 10^{-4}.

Por otra parte, PyToxo se desenvuelve con una razón de aceleración media de 1,90 con respecto a Toxo, con lo que queda patente que también lo mejora en términos de tiempo de ejecución.

Mejorando a Toxo en cuestión de usabilidad, alcance, precisión y tiempo de ejecución, PyToxo constituye una herramienta a tener en cuenta en el ámbito específico que abarca: el cálculo de tablas de penetrancia de modelos de epistasia de alto orden. Así, es perfectamente posible que algún investigador lo termine utilizando para tratar de asociar determinadas variaciones genéticas a enfermedades, pudiendo llegar a órdenes de interacción más complejos y configuraciones paramétricas más realistas de lo que se podía hasta este momento en el estado del arte.

PyToxo está disponible para toda la comunidad científica como *software* libre, pudiéndose adquirir desde el repositorio oficial de Python para paquetes de terceros. Por supuesto, finalmente serán sus futuribles usuarios los que tengan que juzgar hasta qué punto resulta apropiado y útil para sus labores, lo cual constituiría sin duda una métrica más fidedigna a la hora de evaluar este Trabajo de Fin de Grado que las que nosotros podemos presentar a día de hoy. No siendo todavía posible, por el momento no nos queda más que liberar este producto limitándonos a defender nuestra postura de satisfacción para con este proyecto, en base a los argumentos expuestos.

7.2 Relación con la titulación

Para la realización de este Trabajo de Fin de Grado se han puesto en valor diferentes competencias adquiridas durante los estudios de Grado en Ingeniería Informática.

La creación de una nueva herramienta, como lo es PyToxo, implica necesariamente la capacidad para analizar, diseñar y desarrollar aplicaciones de forma robusta y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados para este proceso.

Para un correcto análisis del *software* anterior Toxo, en el que se ha basado la nueva implementación, ha sido necesario evaluar la complejidad computacional inherente al mismo y conocer las peculiaridades algorítmicas que presenta. También sus características relativas al diseño *software*, para posibilitar la construcción de una aplicación bien estructurada, favoreciendo su mantenimiento por parte del mismo y terceros desarrolladores.

Han sido necesarios conocimientos sobre los fundamentos teóricos de los lenguajes de programación para afrontar una traducción manual de código entre dos lenguajes, reescribiendo un programa completo. También conocimientos específicos sobre los lenguajes impli-

cados en este proceso, identificando sus diferencias más importantes para adaptar la nueva implementación a la tecnología elegida.

Para el desarrollo de PyToxo se han empleado las habilidades de programación que se cultivan específica y transversalmente a lo largo de todos los estudios de la titulación, así como nociones sobre marcos de desarrollo para la integración y distribución del producto final en la máquina del usuario.

También ha sido indispensable la capacidad para diseñar y evaluar interfaces persona computador que potencien la usabilidad de la herramienta final.

El proceso de afrontar un proyecto original como el que ha implicado este Trabajo de Fin de Grado también pone en valor la aplicación de los principios, metodologías y ciclos de vida de la ingeniería del *software* que se enseñan durante los estudios de Grado en Ingeniería Informática, para desarrollar y evaluar productos que satisfagan todos los requisitos del usuario final y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener, y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas apropiados.

Resumidamente este proyecto ha supuesto el estudio de un ámbito de aplicación concreto como lo es la bioinformática y su vertiente dedicada a los estudios de asociación genética, el análisis de una librería científica original para portarla de cero a una nueva tecnología, la implementación de la nueva librería como tal atendiendo a los principios de diseño canónicos, la implementación de varias interfaces de usuario que potencien su usabilidad, el desarrollo de diversos tipos de pruebas para contrastar su correcto y completo funcionamiento, la distribución de un *software* terminado y documentado en los canales oficiales dedicados a ello, y por último la redacción de esta memoria. Así pues, este Trabajo de Fin de Grado supone un ejercicio de integración de diferentes capacidades adquiridas durante la formación de Grado en Ingeniería Informática y específicamente de la Mención en Computación del mismo.

En general, este proyecto le ha permitido al estudiante poner en práctica las competencias obtenidas durante toda la titulación, además de servirle como introducción al ámbito de la investigación.

7.3 Trabajo futuro

Aunque PyToxo haya mejorado el alcance en el estado del arte para con los modelos epistáticos que es capaz de utilizar, una línea de trabajo clara podría ser seguir incrementando la complejidad admisible. PyToxo ha sido diseñado muy modularmente y no sería complejo acoplar nuevos resolutores, incluso basados en otras aproximaciones matemáticas.

En el ámbito de la usabilidad, una característica que ya se ha deslizado a lo largo de este manuscrito como interesante sería la de traducir las interfaces a otros idiomas, pues en la actualidad sólo se puede utilizar en inglés. También se podría afrontar la implementación de

interfaces para dispositivos actualmente no contemplados, como por ejemplo aquellos con pantalla táctil.

Por último, pero puede que más interesante, puesto que PyToxo ha sido liberado como *software* libre, se ha abierto la puerta a que cualquier desarrollador o usuario interesado pueda proponer las funcionalidades o características que a su juicio mejoren la herramienta, aunque estas no hayan sido consideradas en el diseño original. Tal vez con la experiencia de su manejo surjan nuevas características con las que servir mejor a su propósito, o incluso nuevos propósitos a los que servir.

Apéndices

Apéndice A

Manual de usuario de PyToxo (versión en castellano)

Este anexo constituye un manual de usuario para la instalación y el uso de PyToxo. En el Anexo B puede consultarse su versión equivalente en inglés.

A.1 Requisitos

Para poder instalar PyToxo es necesario disponer previamente de Python 3.8 o superior. Se recomienda seguir las instrucciones oficiales de la Python Software Foundation [31] para instalar Python 3.8.10, que es la última versión de Python 3.8. La instalación de Python del usuario debe incluir también la herramienta PIP [24].

En este manual se asumirá que el usuario ha completado el proceso de instalación de Python y configurado correctamente la variable PATH de su entorno para que tanto el propio Python, como PIP y los paquetes instalados a raíz del mismo, sean alcanzables desde un terminal de comandos.

PyToxo funciona en cualquiera de sus formas —librería programática, CLI y GUI— en sistemas Linux, Mac OS y Windows. Concretamente ha sido verificado sobre Linux Lubuntu 20, Apple Mac OS Big Sur y Microsoft Windows 10. Sin embargo, sus escasas dependencias hacen de PyToxo una aplicación muy portable que es improbable que presente problemas de compatibilidad incluso en plataformas más antiguas.

Además de lo anterior, si se desea utilizar la interfaz gráfica de PyToxo, es necesario disponer de Tk [44]. Tk ha sido adaptada para funcionar en la mayoría de las variantes de Linux, Mac OS y Windows, y está incluida en la mayoría de las distribuciones modernas de Python para las plataformas anteriores. No ocurre así en algunas distribuciones de Linux, en las que hay que instalar un paquete adicional. Sin embargo, el propio PyToxo detecta esta carencia cuando se intenta lanzar la interfaz, y le sugiere al usuario el comando a emplear para instalar

el módulo, que en sistemas operativos basados en Debian es:

```
sudo apt install python3-tk
```

A.2 Instalación

PyToxo está disponible en PyPI [29], el repositorio oficial de Python, por lo que, una vez completados los pasos de la sección anterior, basta con ejecutar el siguiente comando para instalarlo:

```
pip install pytoxo
```

Hecho esto, habrán quedado disponibles en la máquina del usuario la librería `pytoxo` y los ejecutables `pytoxo_cli` y `pytoxo_gui`.

A.3 Uso como librería de Python

A la hora de utilizar PyToxo, y como es bastante idiosincrásico en la comunidad de Python, la mejor documentación la conforman las propias cabeceras del código fuente, que en PyToxo han sido escritas con gran detalle atendiendo a las convecciones recomendadas oficialmente.

Con propósito ilustrativo, a continuación vamos a abordar el uso de PyToxo como librería de Python por medio de una serie de ejemplos. Esta demostración está disponible en forma de Jupyter Notebook [22] en el repositorio de PyToxo.

Lo primero que tenemos que hacer para utilizar PyToxo es importar la librería:

```
import pytoxo
```

Luego, a partir de un modelo en archivo CSV, podemos generar un objeto `Model` de PyToxo con:

```
modelo = pytoxo.Model(filename="../modelos/additive_3.csv")
```

E inmediatamente después ya podemos generar una tabla de penetrancia utilizando el método apropiado del objeto `modelo`. Este método será `find_max_prevalence_table` o `find_max_heritability_table`, según queramos maximizar la prevalencia o la heredabilidad, respectivamente.

```
# Definimos los parámetros del experimento
mafs = [0.4, 0.4, 0.4]
heredabilidad = 0.85

tabla = modelo.find_max_prevalence_table(mafs=mafs, h=heredabilidad)
```

```
import pytoxo
import numpy

genotipos = ["AABB", "AABb", "AAbb", "AaBB", "AaBb", "Aabb",
    "aaBB", "aaBb", "aabb"]
probabilidades = numpy.array(
    [
        "x",
        "x",
        "x",
        "x",
        "x*(1+y)",
        "x*(1+y)",
        "x",
        "x*(1+y)",
        "x*(1+y)",
    ]
)  # Podemos utilizar tanto listas normales como arrays de NumPy
modelo = pytoxo.Model(
    definitions=genotipos,
    probabilities=probabilidades,
    model_name="otro_modelo",
)
tabla = modelo.find_max_heritability_table(mafs=[0.1] *
    modelo.order, p=0.96)
tabla.print_table()
```

Código A.1: Ejemplo de uso de PyToxo como librería introduciendo el modelo manualmente

El objeto `tabla`, de la clase `PTable`, contiene nuestra tabla de penetrancia. Ahora ya podríamos imprimirla por pantalla o guardarla como un archivo, por ejemplo con:

```
tabla.print_table(format="gametes")
```

Utilizando PyToxo como librería también tenemos la posibilidad de ingresar los datos del modelo epistático original directamente, sin recurrir a un archivo CSV existente. En el Código A.1 presentamos un ejemplo de uso completo que parte de dos listas con los datos del modelo.

A.4 Uso desde la interfaz en línea de comando

Para invocar la interfaz en línea de comando o en inglés *command line interface* (CLI) basta con emplear el comando:

```
pytoxo_cli
```

Podemos resumir el uso de la interfaz en la especificación POSIX [8] siguiente, que desglosamos a continuación de la misma, comenzando por los argumentos opcionales:

```
pytoxo [-h] [--gametes] (--max_prev | --max_her) <model>
    <prev_or_her> <maf> [<maf> ...]
```

- `-h`: usando este argumento opcional desplegamos la ayuda en línea de comando.
- `--gametes`: argumento opcional que implica que la tabla de salida será compuesta en el formato de GAMETES [50]. Si no se hace uso de este argumento, la tabla será conformada como CSV, por defecto.
- `--max_prev` o `--max_her`: empleando el primero se maximizará la prevalencia y empleando el segundo, la heredabilidad. Estas dos opciones son mutuamente excluyentes, es necesario especificar una de las dos, e implica que el último argumento ingresado antes de las MAF será tratado como heredabilidad, si se maximiza la prevalencia; o como prevalencia, si se maximiza la heredabilidad.

Los argumentos opcionales anteriores los podemos distribuir en cualquier posición del comando sin que esto afecte al mismo. Sin embargo, en los que sí que resulta relevante la situación es en los argumentos posicionales, que son todos los que no se correspondan sintácticamente con alguno de los anteriores, y que serán interpretados, en orden, como sigue:

- El primer argumento posicional será la ruta hasta el archivo CSV que contenga el modelo epistático.
- El segundo argumento posicional se corresponderá con la heredabilidad o con la prevalencia a fijar, dependiendo de si se ha definido `--max_prev` o `--max_her`.
- Los argumentos posicionales que vayan a continuación de los anteriores serán interpretados todos ellos como cada una de las MAF, respectivamente. Deben ser especificadas tantas MAF como orden tenga el modelo. Las MAF deben ir una detrás de otra en el comando.

A modo ilustrativo, presentamos en el Código A.2 algunos ejemplos de comandos bien formados que hacen uso de la CLI de PyToxo. Los cuatro primeros casos anteriores son distintas formas de escribir exactamente el mismo experimento.

Una vez que la tabla de penetrancia es calculada desde la interfaz en línea de comando, esta se imprime directamente a la salida estándar configurada en el terminal. Para guardar la tabla en un archivo, se debe seguir cualquiera de los métodos habituales en este tipo de flujos de trabajo, como por ejemplo redireccionando la salida:

```
pytoxo_cli modelos/additive_3.csv --max_prev 0.75 0.5 0.45 0.5 >
    mi_tabla.csv
```

```
pytoxo_cli --gametes --max_prev modelos/additive_4.csv 0.6 0.2 0.3
    0.3 0.4

pytoxo_cli modelos/additive_4.csv --max_prev --gametes 0.6 0.2 0.3
    0.3 0.4

pytoxo_cli modelos/additive_4.csv 0.6 0.2 0.3 0.3 0.4 --max_prev
    --gametes

pytoxo_cli modelos/additive_4.csv 0.6 --max_prev 0.2 0.3 0.3 0.4
    --gametes

pytoxo_cli modelos/threshold_8.csv --max_her 0.88 0.01 0.01 0.01
    0.01 0.01 0.01 0.01 0.01

pytoxo_cli modelos/additive_2.csv --max_her 0.4 0.45 0.5
```

Código A.2: Ejemplos de comandos bien formados que hacen uso de la CLI de PyToxo

A.5 Uso desde la interfaz gráfica de usuario

Para iniciar la GUI basta con emplear el comando:

```
pytoxo_gui
```

En la Figura A.1 hemos marcado los distintos componentes que integran la interfaz gráfica, y que comentamos a continuación:

- **a**: menú contextual de archivo, desde el que cargar un modelo, borrarlo, guardar una tabla de penetrancia como archivo o cerrar la aplicación.
- **b**: menú contextual de ayuda, desde el que se puede acceder a información de utilidad sobre la aplicación.
- **c**: parrilla principal de la pantalla, con la representación del modelo. En el caso de la Figura A.1 contiene también en la tercera columna las penetrancias ya calculadas.
- **d**: textos informativos de apoyo al usuario, que se van actualizando dinámicamente.
- **e**: indicador del estado del programa, que se alterna cuando se inicia el proceso de cálculo de la tabla.
- **f**: desplegable para seleccionar el parámetro a fijar de entre prevalencia y heredabilidad, y espacio para ingresar su valor.
- **g**: espacios para rellenar las MAF a emplear. El número de espacios se adapta dinámicamente al orden del modelo cargado.

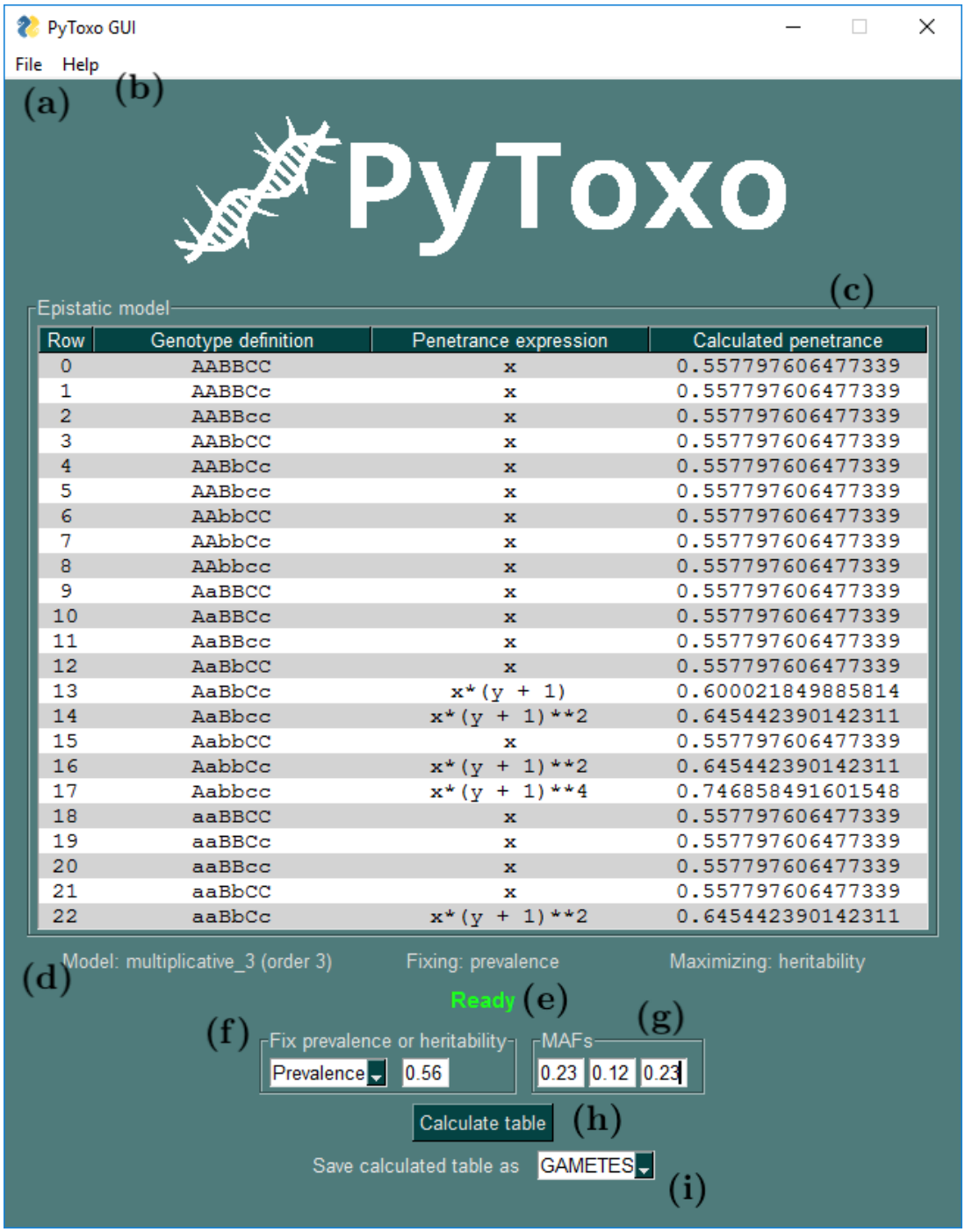

Figura A.1: GUI de PyToxo desde Windows 10 con sus diferentes componentes marcados para referenciar

- **h**: botón para calcular la tabla de penetrancia con la configuración cumplimentada.
- **i**: desplegable para seleccionar el formato de la tabla para guardarla como archivo, de entre CSV y formato de GAMETES [50].

La GUI resulta muy sencilla de utilizar debido a que se adapta dinámicamente al estado en el que se halla el flujo de trabajo de la aplicación. Los botones se van habilitando o deshabilitando en función de si se tiene cargado un modelo, rellenados los campos pertinentes, etc. Por ejemplo, no podemos rellenar las MAF hasta haber cargado un modelo y no podemos usar el botón de calcular hasta que todos los parámetros hayan sido cumplimentados.

Appendix B

PyToxo user manual (English version)

This appendix constitutes an user manual for the installation and use of PyToxo. Its equivalent version in Spanish can be consulted in the Appendix A.

B.1 Requirements

In order to install PyToxo it is necessary to have installed Python 3.8 or higher previously. It is recommended to follow the official instructions of the Python Software Foundation [31] to install Python 3.8.10, which is the latest version of Python 3.8. The user's Python installation must also include the PIP tool [24].

In this manual it will be assumed that the user has completed the Python installation process and correctly configured the PATH variable of their environment, so that both Python itself, as well as PIP and the packages installed as a result of it, are reachable from a command terminal.

PyToxo works in any of its forms —programmatic library, CLI, and GUI— on Linux, Mac OS, and Windows systems. Specifically, it has been verified on Linux Lubuntu 20, Apple Mac OS Big Sur and Microsoft Windows 10. However, its few dependencies make PyToxo a very portable application that is unlikely to present compatibility problems even on older platforms.

In addition to the above, if you want to use the PyToxo graphical interface, you must have Tk [44]. Tk has been adapted to work on most variants of Linux, Mac OS, and Windows, and is included in most modern Python distributions for the older platforms. This is not the case for some Linux distributions, where an additional package must be installed. However, PyToxo itself detects this deficiency when trying to launch the interface, and suggests to the user the command to use to install the module, which in Debian-based operating systems is:

```
sudo apt install python3-tk
```

B.2 Installation

PyToxo is available from PyPI [29], the official Python repository, so once you have completed the steps in the previous section, just run the following command to install it:

```
pip install pytoxo
```

Once this is done, the `pytoxo` library and the executables `pytoxo_cli` and `pytoxo_gui` will be available on the user's machine.

B.3 Use as a Python library

When using PyToxo, and as it is quite idiosyncratic in the Python community, the best documentation is made up of the source code headers themselves, which in PyToxo have been written in great detail according to the officially recommended convections.

For illustrative purposes, below we are going to address the use of PyToxo as a Python library through a series of examples. This demo is available in the form of a Jupyter Notebook [22] in the PyToxo repository.

The first thing we have to do to use PyToxo is to import the library:

```
import pytoxo
```

Then, from a model in a CSV file, we can generate a PyToxo `Model` object with:

```
model = pytoxo.Model(filename="../models/additive_3.csv")
```

And immediately afterwards we can generate a penetrance table using the appropriate method of the `model` object. This method will be `find_max_prevalence_table` or `find_max_heritability_table`, depending on whether we want to maximize prevalence or heritability, respectively.

```
# Definimos los parámetros del experimento
mafs = [0.4, 0.4, 0.4]
heritability = 0.85

table = model.find_max_prevalence_table(mafs=mafs, h=heritability)
```

The object `table`, of class `PTable`, contains our penetrance table. Now we could print it on the screen or save it as a file, for example with:

```
table.print_table(format="gametes")
```

```
import pytoxo
import numpy

genotypes = ["AABB", "AABb", "AAbb", "AaBB", "AaBb", "Aabb",
    "aaBB", "aaBb", "aabb"]
probabilities = numpy.array(
    [
        "x",
        "x",
        "x",
        "x",
        "x*(1+y)",
        "x*(1+y)",
        "x",
        "x*(1+y)",
        "x*(1+y)",
    ]
)  # We also can use Numpy arrays instead of lists
model = pytoxo.Model(
    definitions=genotypes,
    probabilities=probabilities,
    model_name="other_model",
)
table = model.find_max_heritability_table(mafs=[0.1] * model.order,
    p=0.96)
table.print_table()
```

Code B.1: Example using PyToxo as a library and manually entering the model

Using PyToxo as a library we also have the possibility of entering the data of the original epistatic model directly, without using to an existing CSV file. In the Code B.1 is a complete usage example that starts from two lists with the model data.

B.4 Use from the command line interface

To invoke the CLI, just use the command:

```
pytoxo_cli
```

We can summarize the use of the interface in the following POSIX [8] specification, which we break down below, starting with the optional arguments:

```
pytoxo [-h] [--gametes] (--max_prev | --max_her) <model>
    <prev_or_her> <maf> [<maf> ...]
```

- `-h`: using this optional argument we display the command line help.
- `--gametes`: optional argument that implies that the output table will be composed in

the format of GAMETES [50]. If this argument is not used, the table will be configured as CSV, by default.

- `--max_prev` or `--max_her`: using the first will maximize the prevalence and using the second, the heritability. These options are mutually exclusive, specifying one is mandatory, and implies that the last argument entered before the MAF will be treated as heritability, if prevalence is maximized; or as prevalence, if heritability is maximized.

The previous optional arguments can be distributed in any position of the command without affecting it. However, positional arguments, or arguments that do not match with any of the previously defined options, are interpreted according to the order in which they are provided:

- The first positional argument will be the path to the CSV file containing the epistatic model.
- The second positional argument will correspond to the heritability or the prevalence to be set, depending on what has been selected with `--max_prev` or `--max_her`.
- The positional arguments that follow the previous ones will all be interpreted as each of the MAFs, respectively. The number of MAFs provided has to match the order of the model previously introduced. MAFs are separated by spaces, just as any other argument.

In the Code B.2 are some examples of well-formed commands that make use of the PyToxo CLI. The first four cases above are different ways of writing the exact same experiment.

Once the penetrance table is calculated from the command line interface, it is printed directly to the standard output configured in the terminal. To save the table to a file, you must follow any of the usual methods in this type of workflow, such as redirecting the output of the command:

```
pytoxo_cli models/additive_3.csv --max_prev 0.75 0.5 0.45 0.5 >
    my_table.csv
```

B.5 Use from the graphical user interface

To start the GUI, just use the command:

```
pytoxo_gui
```

In Figure B.1 we have marked the different components that make up the graphical interface, and we comment on them below:

```
pytoxo_cli --gametes --max_prev models/additive_4.csv 0.6 0.2 0.3
    0.3 0.4

pytoxo_cli models/additive_4.csv --max_prev --gametes 0.6 0.2 0.3
    0.3 0.4

pytoxo_cli models/additive_4.csv 0.6 0.2 0.3 0.3 0.4 --max_prev
    --gametes

pytoxo_cli models/additive_4.csv 0.6 --max_prev 0.2 0.3 0.3 0.4
    --gametes

pytoxo_cli models/threshold_8.csv --max_her 0.88 0.01 0.01 0.01
    0.01 0.01 0.01 0.01 0.01

pytoxo_cli models/additive_2.csv --max_her 0.4 0.45 0.5
```

Code B.2: Examples of well-formed commands that make use of the PyToxo CLI

- **a**: file contextual menu, from which to load a model, delete it, save a penetrance table as a file or close the application.
- **b**: contextual help menu, from which useful information about the application can be accessed.
- **c**: main grid of the screen, with the representation of the model. In the case of Figure B.1 it also contains the penetrations already calculated in the third column.
- **d**: informative texts to support the user, which are dynamically updated.
- **e**: program status indicator, which toggles when the table calculation process starts.
- **f**: drop-down to select the parameter to be set between prevalence and heritability, and space to enter its value.
- **g**: spaces to fill in the MAF to use. The number of spaces is dynamically adapted to the order of the loaded model.
- **h**: button to calculate the penetrance table with the completed configuration.
- **i**: drop-down to select the table format to save it as a file, from CSV and GAMETES [50] format.

The GUI is very easy to use because it dynamically adapts to the state of the application workflow. The buttons are enabled or disabled depending on whether a model is loaded, the relevant fields filled in, etc. For example, we cannot fill in MAFs until we have loaded a model and we cannot use the calculate button until all parameters have been filled.

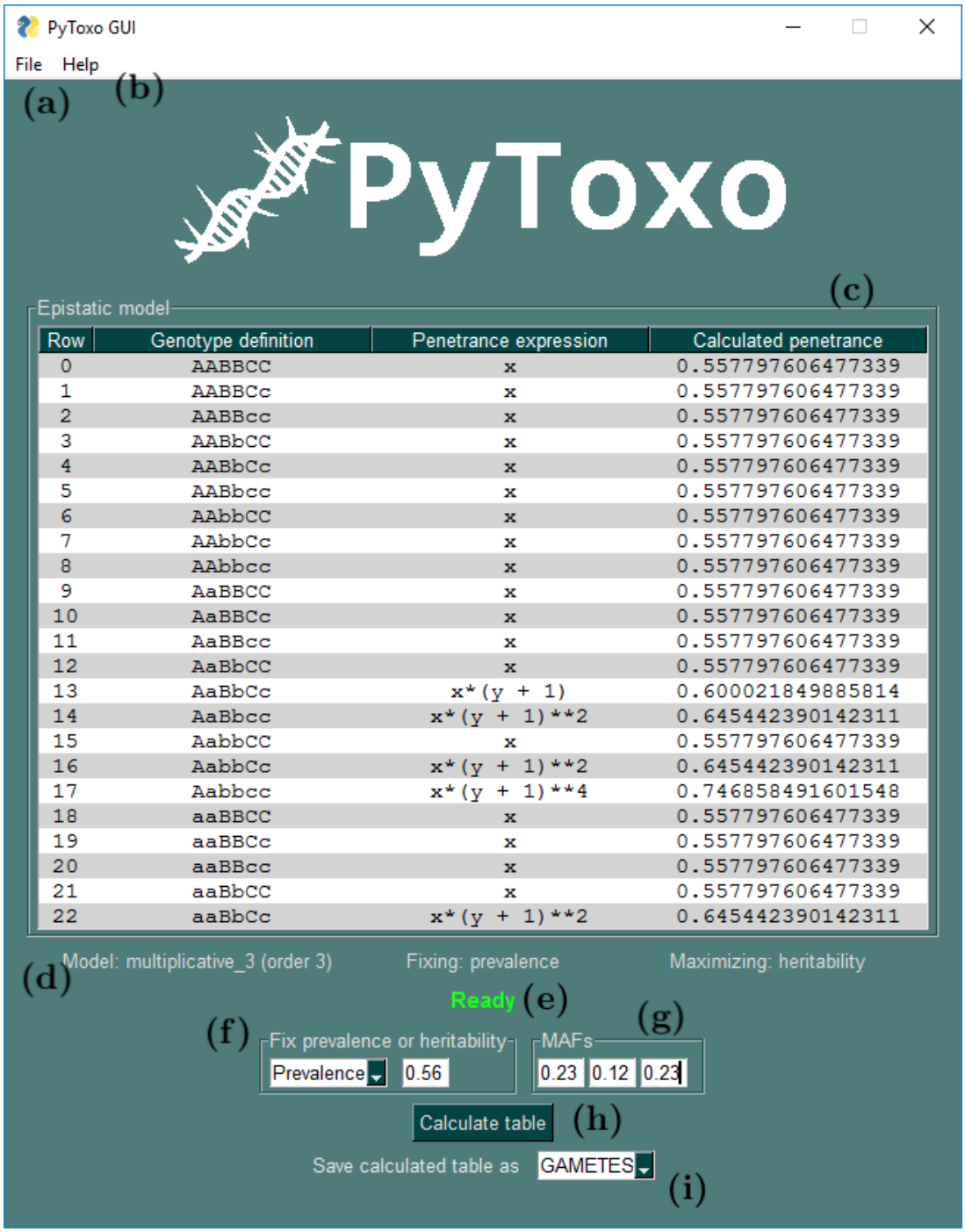

Figure B.1: PyToxo GUI from Windows 10 with its different components marked for reference

Lista de acrónimos

CLI interfaz en línea de comando o en inglés *command line interface*. 31, 50, 69, 71, 77, 79

CSV valores separados por comas o en inglés *comma-separated values*. 23, 31, 44, 70, 72, 75, 78, 80, 81

GUI interfaz gráfica de usuario o en inglés *graphical user interface*. 31, 32, 50, 69, 73, 77, 80

GWAS estudios de asociación del genoma completo o en inglés *genome-wide association studies*. 1, 6, 45, 63

MAF frecuencia del alelo menos común o en inglés *minor allele frequency*. 8, 23, 28, 31, 33, 44, 49, 72, 73, 80, 81

SNP polimorfismos de un solo nucleótido o en inglés *single nuclotide polymorphisms*. 1

Glosario

alelo cada una de las dos o más versiones de un determinado gen. 2, 5, 6, 8, 44

epistasia (sin. *epistasis*) fenómeno por el que la expresión de un gen se ve afectada por la expresión de uno o más genes heredados de forma independiente. 1, 6, 43, 64

fenotipo conjunto de rasgos observables de un organismo que se manifiestan como resultado de la interacción entre su genotipo y el medio ambiente circundante. 1, 5, 6

genotipo colección de genes de un individuo. La expresión del genotipo contribuye a los caracteres observables del mismo (fenotipo). 1, 6

heredabilidad cantidad de variación fenotípica que corresponde a la variación genética. 6, 8, 23, 26, 28, 31, 33, 44, 48, 49, 70, 72, 73

heterocigoto describe el genotipo de un organismo diploide en un locus específico de su ADN, de forma que para un gen en particular estarían presentes alelos diferentes para ese gen en ambos cromosomas homólogos. 44

homocigoto describe el genotipo de un organismo diploide en un locus específico de su ADN, de forma que para un gen en particular estarían presentes alelos idénticos para ese gen en ambos cromosomas homólogos. 44

locus (pl. *loci*) lugar específico del cromosoma donde está localizado un gen u otra secuencia de ADN. 1, 6, 8, 44

prevalencia proporción de individuos en una población que posee el fenotipo objeto de estudio. 6, 8, 19, 26, 28, 31, 33, 44, 48, 70, 72, 73

Bibliografía

[1] X. Cao, G. Yu, J. Liu, L. Jia, y J. Wang, «ClusterMI: Detecting High-Order SNP Interactions Based on Clustering and Mutual Information», *International Journal of Molecular Sciences*, vol. 19, n.º 8, 2018. DOI: 10.3390/ijms19082267.

[2] S. Cass, «The top programming languages: Our latest rankings put Python on top-again - [Careers]», *IEEE Spectrum*, vol. 57, n.º 8, p. 22, 2020. DOI: 10.1109/MSPEC.2020.9150550.

[3] B. Chapman y J. Chang, «Biopython: Python Tools for Computational Biology», *SIGBIO Newsl.*, vol. 20, n.º 2, pp. 15–19, 2000. DOI: 10.1145/360262.360268.

[4] G. Churchill, «Epistasis», en *Brenner's Encyclopedia of Genetics (Second Edition)*, S. Maloy y K. Hughes, Eds. San Diego: Academic Press, 2013, p. 505. DOI: 10.1016/B978-0-12-374984-0.00482-4.

[5] T. L. Edwards, W. S. Bush, S. D. Turner, S. M. Dudek, E. S. Torstenson, M. Schmidt, E. Martin, y M. D. Ritchie, «Generating Linkage Disequilibrium Patterns in Data Simulations Using genomeSIMLA», en *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, E. Marchiori y J. H. Moore, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 24–35. DOI: 10.1007/978-3-540-78757-0_3.

[6] B. González Seoane, «PyToxo en GitHub», https://github.com/bglezseoane/pytoxo, 2021, consultado el 21 de junio de 2021.

[7] J. Hartley, «Colorama en GitHub», https://github.com/tartley/colorama, consultado el 3 de junio de 2021.

[8] IEEE Computer Society y The Open Group, «IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7», *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, 2018. DOI: 10.1109/IEEESTD.2018.8277153.

[9] JetBrains S.R.O., «PyCharm: the Python IDE for Professional Developers», https://www.jetbrains.com/pycharm/, consultado el 16 de junio de 2021.

[10] F. Johansson *et al.*, «mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.2.1)», http://mpmath.org/, 2013, consultado el 31 de mayo de 2021.

[11] K. Lepa, «*termcolor* en PyPI», https://pypi.org/project/termcolor/, consultado el 31 de mayo de 2021.

[12] F. Lundh, A. Clark *et al.*, «Pillow», https://python-pillow.org, consultado el 31 de mayo de 2021.

[13] J. Marchini, P. Donnelly, y L. R. Cardon, «Genome-wide strategies for detecting multiple loci that influence complex diseases», *Nature Genetics*, vol. 37, n.º 4, pp. 413–417, 2005. DOI: 10.1038/ng1537.

[14] D. Mariano, M. Ferreira, B. L. Sousa, L. H. Santos, y R. C. de Melo-Minardi, «A Brief History of Bioinformatics Told by Data Visualization», en *Advances in Bioinformatics and Computational Biology*, J. C. Setubal y W. M. Silva, Eds. Springer International Publishing, 2020, pp. 235–246. DOI: 10.1007/978-3-030-65775-8_22.

[15] R. J. Neuman, J. P. Rice, y A. Chakravarti, «Two-Locus models of disease: Two-Locus Models of Disease», *Genetic Epidemiology*, vol. 9, n.º 5, pp. 347–365, 1992. DOI: 10.1002/gepi.1370090506.

[16] C. Niel, C. Sinoquet, C. Dina, y G. Rocheleau, «SMMB: a stochastic Markov blanket framework strategy for epistasis detection in GWAS», *Bioinformatics*, vol. 34, n.º 16, pp. 2773–2780, 2018. DOI: 10.1093/bioinformatics/bty154.

[17] NumPy, «NumPy: the fundamental package for scientific computing with Python», https://numpy.org, consultado el 27 de mayo de 2021.

[18] B. Peng y M. Kimmel, «simuPOP: a forward-time population genetics simulation environment», *Bioinformatics*, vol. 21, n.º 18, pp. 3686–3687, 2005. DOI: 10.1093/bioinformatics/bti584.

[19] Pierre Carbonnelle, «PYPL PopularitY of Programming Language», https://pypl.github.io, consultado el 26 de mayo de 2021.

[20] C. Ponte-Fernández, «Toxo en GitHub», https://github.com/UDC-GAC/toxo, 2020, consultado el 26 de mayo de 2021.

[21] C. Ponte-Fernández, J. González-Domínguez, A. Carvajal-Rodríguez, y M. J. Martín, «Toxo: a library for calculating penetrance tables of high-order epistasis models», *BMC Bioinformatics*, vol. 21, n.º 1, p. 138, 2020. DOI: 10.1186/s12859-020-3456-3.

[22] Project Jupyter, «Jupyter», https://jupyter.org, consultado el 16 de junio de 2021.

[23] PySimpleGUI, «PySimpleGUI en GitHub», https://github.com/PySimpleGUI/PySimpleGUI, consultado el 31 de mayo de 2021.

[24] Python Packaging Authority, «PIP Documentation v21.1.2», https://pip.pypa.io/en/stable/, consultado el 3 de junio de 2021.

[25] ——, «Setuptools en GitHub», https://github.com/pypa/setuptools, consultado el 3 de junio de 2021.

[26] Python Software Foundation, «Parser for command-line options, arguments and subcommands», https://docs.python.org/3/library/argparse.html, consultado el 16 de junio de 2021.

[27] ——, «Docstring conventions», https://www.python.org/dev/peps/pep-0257/, consultado el 11 de junio de 2021.

[28] ——, «Style guide for python code», https://www.python.org/dev/peps/pep-0008/, consultado el 11 de junio de 2021.

[29] ——, «PyPI», https://pypi.org, consultado el 28 de mayo de 2021.

[30] ——, «Python», https://www.python.org, consultado el 26 de mayo de 2021.

[31] ——, «Python 3.8.10», https://www.python.org/downloads/release/python-3810/, consultado el 31 de mayo de 2021.

[32] ——, «Graphical user interfaces with Tk», https://docs.python.org/3/library/tk.html, consultado el 3 de junio de 2021.

[33] ——, «Support for type hints», https://docs.python.org/3/library/typing.html, consultado el 11 de junio de 2021.

[34] ——, «Unit testing framework», https://docs.python.org/3/library/unittest.html, consultado el 4 de junio de 2021.

[35] M. Pérez-Enciso, N. Forneris, G. de los Campos, y A. Legarra, «Evaluating Sequence-Based Genomic Prediction with an Efficient New Simulator», *Genetics*, vol. 205, n.º 2, p. 939, febr. 2017. DOI: 10.1534/genetics.116.194878.

[36] Qt Project, «Qt», https://www.qt.io/product, consultado el 9 de junio de 2021.

[37] SciPy Developers, «SciPy», https://www.scipy.org, consultado el 27 de mayo de 2021.

[38] J. Shang, J. Zhang, X. Lei, W. Zhao, y Y. Dong, «EpiSIM: simulation of multiple epistasis, linkage disequilibrium patterns and haplotype blocks for genome-wide interaction analysis», *Genes & Genomics*, vol. 35, n.º 3, pp. 305–316, jun. 2013. DOI: 10.1007/s13258-013-0081-9.

[39] Software Freedom Conservancy, Inc., «Git», https://git-scm.com, consultado el 16 de junio de 2021.

[40] Stack Overflow, «Stack Overflow Survey 2020», https://insights.stackoverflow.com/survey/2020, 2020, consultado el 26 de mayo de 2021.

[41] SymPy Development Team, «SymPy», https://www.sympy.org/en/index.html, consultado el 27 de mayo de 2021.

[42] ——, «Solvers», https://docs.sympy.org/latest/tutorial/solvers.html, consultado el 7 de junio de 2021.

[43] M. B. Taylor y I. M. Ehrenreich, «Higher-order genetic interactions and their contribution to complex traits», *Trends in Genetics*, vol. 31, n.º 1, pp. 34–40, 2015. DOI: 10.1016/j.tig.2014.09.001.

[44] Tcl Core Team, «Tcl Developer Xchange», http://www.tcl.tk, consultado el 9 de junio de 2021.

[45] The Computer Language Benchmarks Game, «The Computer Language Benchmarks Game», https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gcc.html, consultado el 28 de mayo de 2021.

[46] The MathWorks, Inc., «MATLAB», https://es.mathworks.com/products/matlab.html, consultado el 28 de mayo de 2021.

[47] The SciPy community, «A Guide to NumPy Dotcumentation», https://numpy.org/doc/stable/docs/howto_document.html, consultado el 11 de junio de 2021.

[48] The UCLEB Consortium, D. Speed, N. Cai, M. R. Johnson, S. Nejentsev, y D. J. Balding, «Reevaluation of SNP heritability in complex human traits», *Nature Genetics*, vol. 49, n.º 7, pp. 986–992, jul. 2017. DOI: 10.1038/ng.3865.

[49] TIOBE Software, «TIOBE Index for June 2021», https://www.tiobe.com/tiobe-index/, consultado el 21 de junio de 2021.

[50] R. J. Urbanowicz, J. Kiralis, N. A. Sinnott-Armstrong, T. Heberling, J. M. Fisher, y J. H. Moore, «GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures», *BioData Mining*, vol. 5, n.º 1, p. 16, 2012. DOI: 10.1186/1756-0381-5-16.

[51] F. A. Wright, H. Huang, X. Guan, K. Gamiel, C. Jeffries, W. T. Barry, F. Pardo-Manuel de Villena, P. F. Sullivan, K. C. Wilhelmsen, y F. Zou, «Simulating association studies: a data-based resampling method for candidate regions or whole genome scans», *Bioinformatics*, vol. 23, n.º 19, pp. 2581–2588, oct. 2007. DOI: 10.1093/bioinformatics/btm386.

[52] C.-H. Yang, L.-Y. Chuang, y Y.-D. Lin, «Multiobjective multifactor dimensionality reduction to detect SNP–SNP interactions», *Bioinformatics*, vol. 34, n.º 13, pp. 2228–2236, jul. 2018. DOI: 10.1093/bioinformatics/bty076.

[53] Łukasz Langa *et al.*, «Black en GitHub», https://github.com/psf/black, consultado el 3 de junio de 2021.