Algoritmos Genéticos con PYTHON

Un enfoque práctico para resolver problemas de ingeniería

Daniel Gutiérrez Reina
Alejandro Tapia Córdoba
Álvaro Rodríguez del Nozal

Marcombo 1945-2020 75 años

Algoritmos Genéticos con Python

Un enfoque práctico para resolver problemas de ingeniería

Primera edición

Algoritmos Genéticos con Python

Un enfoque práctico para resolver problemas de ingeniería

Primera edición

Algoritmos Genéticos con Python
Un enfoque práctico para resolver problemas de ingeniería

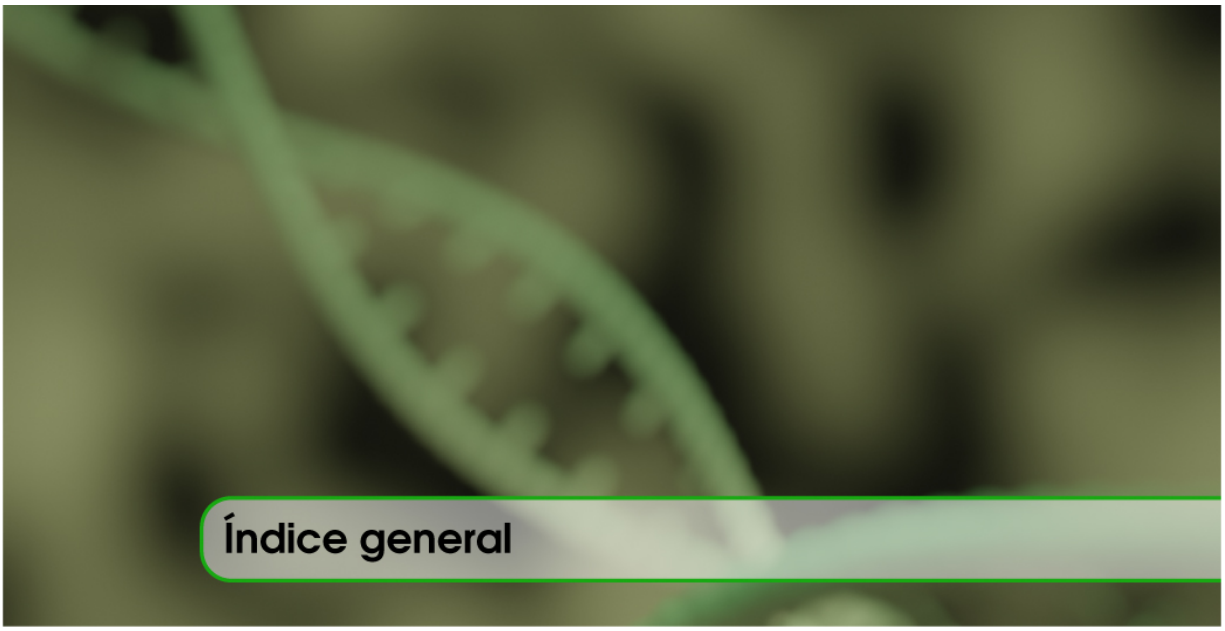
Primera edición, 2020

www.marcombo.com

Diseño de la cubierta: Alejandro Tapia Córdoba
Diseño de interior: Daniel Gutiérrez Reina, Alejandro Tapia Córdoba y Alvaro Rodríguez del Nozal
Correctora: Anna Alberola
Directora de producción: M.ª Rosa Castillo

ISBN: 978-84-267-3068-8
Producción del ebook: booqlab

A Mónica, Laura, Sol y Lucía.



Índice general

¿Por qué algoritmos genéticos en el ámbito de la ingeniería?

Los ingenieros nos enfrentamos día a día a numerosos problemas. La complejidad de estos problemas crece de una forma exponencial debido a las herramientas informáticas que nos permiten desarrollar modelos más complejos. Hemos pasado, pues, de una ingeniería con papel, lápiz y calculadora, a otra con herramientas digitales, gran cantidad de datos (*big data*) y súper ordenadores. Todo ello ha sido posible gracias a los avances conseguidos en disciplinas como la tecnología electrónica, el desarrollo *software* e Internet. En la actualidad, los modelos con los que nos enfrentamos son más realistas, pero también son más difíciles de analizar mediante técnicas analíticas o exactas. En los últimos años, los algoritmos de inteligencia artificial se han convertido en una herramienta indispensable para resolver problemas de ingeniería de una manera aproximada. Estos algoritmos están soportados por la gran capacidad de cálculo que tienen nuestros ordenadores. En la actualidad, cualquier ordenador personal o portátil es capaz de realizar millones de operaciones por segundo. Por otro

lado, lenguajes de programación de alto nivel, como *Python*, han generado una gran cantidad de usuarios, agrupados en comunidades, que trabajan conjuntamente para desarrollar un amplio abanico de recursos abiertos. Estos recursos se engloban en librerías, repositorios abiertos, congresos nacionales e internacionales, comunidades locales, seminarios, etc. El resultado es una democratización de la inteligencia artificial abierta y al alcance de todos. Es decir, las técnicas de inteligencia artificial están a disposición de cualquiera que tenga unos conocimientos medios en matemáticas y programación. Así, los ingenieros -como grandes artífices del desarrollo tecnológico- debemos estar en la cresta de la ola de la inteligencia artificial, para seguir aportando soluciones a los problemas que nos surgen día a día.

En esta gran caja de herramientas compuesta por todos los algoritmos de inteligencia artificial -tales como algoritmos de *machine learning*, redes neuronales, etc.- los algoritmos genéticos son una herramienta indispensable, ya que nos permiten obtener soluciones adecuadas a problemas muy complejos que no se pueden abordar con métodos clásicos. Por lo tanto, cualquiera que se considere experto en la materia debe dominar la técnica. Siempre debemos tener presente que lo importante de cada método es saber en qué escenarios se debe utilizar, qué ventajas tiene y cuáles son las limitaciones de cada herramienta. En este libro se muestran, mediante una serie de ejemplos prácticos, las bondades y las limitaciones de los algoritmos genéticos para resolver problemas de ingeniería.

Como introducción al contenido del libro, debemos anticipar que los algoritmos genéticos se basan en la naturaleza. Esto significa que muchos de los problemas de ingeniería se pueden resolver, simplemente, observando cómo funcionan los seres vivos. El mecanismo utilizado no es otro que la Teoría de la Evolución de Charles Darwin (Darwin, 1859), la cual nos dice que los individuos que mejor se adaptan al medio tienen más probabilidades de sobrevivir y en consecuencia, de dejar descendencia. Esta idea, aparentemente tan alejada del mundo de la ingeniería, ha dado lugar a una metodología de optimización de problemas: los algoritmos evolutivos o

computación evolutiva, donde se encuentran enmarcados los algoritmos genéticos.

A lo largo del libro se abordarán distintos problemas de optimización que se resolverán mediante algoritmos genéticos. El objetivo principal del libro es dejar patente la gran capacidad que tienen los algoritmos genéticos como técnica de resolución de problemas de ingeniería. Así, esperamos que este libro sirva para que muchos ingenieros se introduzcan dentro del mundo de la optimización metaheurística y la apliquen en sus problemas en el futuro.

Daniel Gutiérrez Reina, Alejandro Tapia Córdoba, Álvaro Rodríguez del Nozal

Sevilla, mayo de 2020

Objetivos y estructura del libro

Este libro pretende ofrecer una visión general sobre el desarrollo y la programación de algoritmos genéticos desde un punto de vista práctico y con un enfoque orientado a la resolución de problemas de ingeniería. El libro se ha orientado a un aprendizaje mediante ejemplos (*learning by doing*). Esto significa que los conceptos se van describiendo conforme aparecen en el problema que se aborda. Por lo tanto, no existe un capítulo donde se encuentren todos los operadores genéticos, o las implementaciones de algoritmos genéticos, etc. No obstante, el glosario del libro permite identificar fácilmente la página donde se encuentra cada concepto.

El libro se estructura en dos partes bien diferenciadas. En la primera parte, se cubren los conceptos básicos de los algoritmos genéticos mediante varios ejemplos clásicos. En el primero de los cuatro capítulos que constituyen esta primera parte, se resuelve un problema muy sencillo formulado con variables continuas, con el propósito de ilustrar los principales componentes de los algoritmos genéticos. Aunque el primer capítulo es largo, es necesario leerlo con detenimiento para poder

comprender los principales mecanismos que hay detrás de un algoritmo genético. Por lo tanto, se recomienda no avanzar si no se tienen claros los conceptos descritos en este capítulo. En el segundo capítulo, se aborda el problema del viajero o *Traveling Salesman Problem (TSP)*, sin duda uno de los problemas combinatorios clásicos con variables discretas más estudiados, y que constituye un ejemplo perfecto para demostrar la potencialidad de los algoritmos genéticos para resolver problemas complejos. Una vez conocida la estructura fundamental de los algoritmos genéticos, en el tercer capítulo, se profundiza en el uso de las funciones de *benchmark* para validar tanto sus capacidades como sus potenciales vulnerabilidades, lo cual la constituye una herramienta fundamental para su depuración. Las funciones de *benchmark* son funciones que la comunidad científica utiliza para la evaluación de algoritmos de optimización. Estas presentan diversas dificultades a los algoritmos de optimización. Por ejemplo, tener varios máximos o mínimos (funciones multimodales), o tener un máximo/mínimo local cerca del absoluto. Por último, en el cuarto capítulo se introduce el enfoque multiobjetivo de los algoritmos genéticos, lo cual constituye una de las capacidades más interesantes y versátiles de este tipo de algoritmos. Desde el punto de vista de los problemas de ingeniería, el enfoque multiobjetivo es muy importante, ya que los ingenieros siempre debemos tener en cuenta una relación de compromiso entre el coste y la adecuación de las soluciones al problema. Los problemas multiobjetivo se abordarán mediante dos ejemplos. En primer lugar, se resolverá un problema clásico como es la suma de subconjuntos. Y en segundo lugar, se usarán funciones de *benchmark* con múltiples objetivos. Al finalizar la primera parte, el lector habrá adquirido suficiente destreza como para poder abordar problemas de optimización mediante algoritmos genéticos.

En la segunda parte, se introducirán una serie de problemas ingenieriles, cuya resolución se abordará mediante el desarrollo de algoritmos genéticos. Todos los problemas se tratan tanto desde el punto de vista de un único objetivo (problemas unimodales) como desde el punto de vista de un multiobjetivo (problemas multimodales). En el primer capítulo, se estudia el problema del despacho económico de una microrred eléctrica. Este

problema, formulado en variables continuas, constituye uno de los problemas más complejos y de más relevancia en el área de sistemas eléctricos de potencia, y persigue la programación de la potencia suministrada por un conjunto de generadores, para abastecer una demanda durante un periodo determinado y de forma óptima. En el segundo capítulo, se aborda un problema de optimización relativo al diseño de una planta micro-hidráulica. Este problema, formulado con variables binarias, persigue determinar el trazado óptimo de la planta, y constituye un problema de especial interés dado el alto número de combinaciones posibles, lo que hace inabordable su resolución mediante estrategias analíticas o exactas. Por último, en el tercer capítulo se aborda el problema del posicionamiento óptimo de sensores, en el cual se persigue determinar las posiciones más adecuadas para instalar una serie de sensores de manera que la mayor parte posible de puntos de interés queden cubiertos.

En todos los capítulos se incluye una sección de código completo, lecciones aprendidas y ejercicios propuestos. Las lecciones aprendidas hacen referencias a los aspecto más relevantes que se deben adquirir en dicho capítulo. Los ejercicios sirven para afianzar conceptos y coger destreza en la aplicación de algoritmos genéticos. Por último, para finalizar cada capítulo se incluye una sección con bibliografía adicional para seguir profundizando en los temas abordados en el capítulo.

Prerrequisitos para seguir el libro

Para seguir correctamente el libro, se presupone unos conocimientos medios del lenguaje de programación *Python*. Este libro no cubre los conceptos básicos de este lenguaje, y da por hecho que el lector parte con conocimientos básicos de programación orientada a objetos.

En cuanto a los algoritmos genéticos, el libro cubre desde cero, y paso a paso, los conceptos básicos de dichos algoritmos -tanto de aquellos con un único objetivo, como de aquellos con múltiples objetivos-. El contenido matemático del libro es mínimo, y únicamente es de relevancia en la

segunda parte, donde algunas ecuaciones son necesarias para plantear los problemas de ingeniería propuestos.

Se recomienda una lectura en profundidad de los primeros capítulos antes de pasar a la segunda parte del libro. En la segunda parte se pasa más rápidamente por los componentes de los algoritmos genéticos que se han detallado en la primera parte.

Código

Descripción del código

La presentación del código en todos los capítulos se hace mediante el siguiente procedimiento:

- En primer lugar, se describen por separado cada una de las partes del código utilizadas para resolver los problemas planteados en cada capítulo. De esta forma, se describen paso a paso los principales componentes del algoritmo.
- En segundo lugar, todos los capítulos tienen una sección que incluye el código completo necesario para resolver el problema. Así, el lector puede ver de manera conjunta todas las líneas de código, junto a una breve descripción del mismo.

Para desarrollar el código se ha utilizado el paquete *Anaconda* con *Python 3*. Este paquete incluye tanto el intérprete de *Python* como librerías básicas de este lenguaje de programación, como pueden ser *numpy* o *matplotlib*. Además, nos provee de entornos de desarrollo para generar nuestro código, como pueden ser *Spyder* o *Jupyter*.

Repositorio

Todos los *scripts* utilizados en cada una de las secciones, así como diverso material complementario del libro, se pueden encontrar en el siguiente repositorio de Github:

https://github.com/Dany503/Algoritmos-Geneticos-en-Python-Un-Enfoque-Practico.

Descripción del código

Todos los fragmentos de código desarrollados en este libro se clasifican en cuatro categorías: archivos de texto, resultados, *scripts* y códigos completos. Para facilitar su identificación, cada una de estas categorías se corresponde con un color:

```
Archivo de texto:
Aquí se mostrará el contenido de archivos con los que trabajaremos, bien
como entradas para nuestros algoritmos (por ejemplo, bancos de datos) o
bien como salidas (por ejemplo, un listado con las soluciones de
nuestro problema).
```

```
Resultado:
Los resultados muestran operaciones realizadas directamente en la consola
de Python, y muestran interacciones de tipo entrada-salida.
```

```
Script:
Los scripts sirven para mostrar comandos o fragmentos de código en Python.
A lo largo de este libro utilizaremos scripts para mostrar, por ejemplo,
cómo llamar a ciertas funciones, cómo operar con variables o cómo definir
los operadores que usaremos en nuestros algoritmos genéticos. El código
completo utilizado en cada sección se pueden conseguir mediante la únión
de cada uno de los scripts utilizados en dicha sección.
```

```
1 Código completo:
2 Los códigos completos representan la versión final de nuestros algoritmos,
3 de forma que puedan ser ejecutados de forma autónoma y sin requerir ningún
4 comando adicional. Como regla general, al final de cada capítulo veremos
5 un código completo que permitirá poner en práctica todo lo aprendido. Además,
6 este código está numerado, de manera que podremos describir cada parte,
7 para una mejor comprensión.
```

Algoritmos y operadores de referencia

Aunque las herramientas fundamentales de los algoritmos genéticos se introducirán en el primer capítulo, a lo largo del libro se irán presentando diferentes propuestas para su implementación, en base a las necesidades del problema en estudio. Junto con cada nueva propuesta se presentará una descripción detallada en un entorno como el siguiente:

Operador [Función nueva: *nombre(argumento1,argumento2)*[a]] Aquí se detallará la finalidad de la función y cómo se utiliza, describiendo cada argumento de entrada y cada salida.

[a]Cuando la función presentada esté incluida en la librería de *deap*, se incluirá una referencia a la documentación.

Así, utilizaremos este entorno para describir operadores genéticos (mutación, cruce y selección) y algoritmos genéticos. Para localizar dónde se describe una función en particular, podemos consultar el glosario.

Librerías necesarias

Todos los *scripts* de *Python* se han desarrollado en Anaconda[1]. La versión de *Python* utilizada es 3.6 para Windows. No obstante, no debe haber problemas con otras versiones de *Python* y otros sistemas operativos.

A continuación, se listan las librerías utilizadas:

- *deap*: versión 1.3.
- *matplotlib*: versión 3.1.3
- *numpy*: versión 1.16.3
- *scipy*: versión 1.2.1
- *scoop*: versión 0.7
- Módulos nativos de *Python* como *random*, *arrays*, *multiprocessing*, *JSON*, *math*, etc.

Para instalar la librería *deap* con *pip*[2]:

```
>>> pip install deap
```

Si realiza la instalación desde *Spyder* o Google colab:

```
>>> !pip install deap
```

Para instalar con *conda*[3]:

```
>>> conda install -c conda-forge deap
```

Agradecimientos

Los autores quieren transmitir sus agradecimientos a la Universidad de Sevilla y a la Universidad Loyola Andalucía, instituciones donde actualmente trabajan. Los autores agradecen a todos los desarrolladores de la librería *deap* (Fortin et al., 2012) por la documentación disponible y el esfuerzo desarrollado en los últimos años. Por último, agradecer a compañeros de trabajo, familiares y amigos, por su apoyo.

Sobre los autores y datos de contacto

Daniel Gutiérrez Reina es Doctor Ingeniero en Electrónica por la Universidad de Sevilla (2015). Trabaja actualmente como investigador postdoctoral en el Departamento de Ingeniería Electrónica de la Universidad de Sevilla. Ha sido investigador visitante en la John Moores University (Reino Unido), en Freie Universität Berlin (Alemania), en Colorado School of Mines (Estados Unidos) y en Leeds Beckett University (Reino Unido). También trabajó en la Universidad Loyola Andalucía en el Departamento de Ingeniería durante año y medio. Su investigación se centra en la optimización de problemas de ingeniería utilizando técnicas de optimización metaheurísticas y *machine learning*. Es docente de un gran número de cursos de *Python*, optimización y *machine learning* en la Universidad de Sevilla, en la Universidad de Málaga y en la Universidad de Córdoba. Para contactar con el autor: `dgutierrezreina@us.es`.

Alejandro Tapia Córdoba es Ingeniero Industrial especializado en Materiales (2014) por la Universidad de Sevilla. En 2019 recibió su título de Doctor en Ciencia de los Datos por la Universidad Loyola Andalucía, donde actualmente trabaja como profesor asistente en el Departamento de Ingeniería. Ha sido investigador visitante en la Universidad de Greenwich (UK). Su investigación se enmarca en el desarrollo de estrategias de optimización para aplicaciones en diferentes áreas de la ingeniería. Para contactar con el autor: `atapia@uloyola.es`.

Álvaro Rodríguez del Nozal es Ingeniero Industrial especializado en Automática Industrial (2013) y Máster en Sistemas de Energía Eléctrica (2016) por la Universidad de Sevilla. Recibió su título de Doctor en Ingeniería de Control por la Universidad Loyola Andalucía en el año 2019. Actualmente trabaja como investigador postdoctoral en el Departamento de Ingeniería Eléctrica de la Universidad de Sevilla. Ha sido investigador visitante en el Laboratoire d'analyse et d'architecture des systèmes (Francia) y en el Politecnico di Milano (Italia). Su investigación se centra en el control y estimación distribuida de sistemas dinámicos, así como en la integración de energías renovables en la red eléctrica. Para contactar con el autor: `arnozal@us.es`.

[1]Se puede descargar de forma gratuita en https://www.anaconda.com/distribution/

[2]https://pypi.org/project/deap/

[3]https://anaconda.org/conda-forge/deap

Parte 1: Introducción a los algoritmos genéticos

1.1 Introducción a los algoritmos genéticos

Los algoritmos genéticos son técnicas de optimización metaheurísticas, también llamadas estocásticas o probabilísticas (Holland et al., 1992) (Goldberg, 2006). Aunque fueron propuestos en la década de los 60s por Jonh Holland (Holland, 1962) (Holland, 1965) (Holland et al., 1992), no ha sido posible su aplicación en problemas de ingeniería reales hasta hace un par de décadas, debido principalmente a que son computacionalmente intensivos y que, por lo tanto, necesitan una capacidad computacional elevada para llevar a cabo multitud de operaciones en poco tiempo. La idea principal de un algoritmo genético es realizar una búsqueda guiada a partir de un conjunto inicial de posibles soluciones, denominado población inicial, el cual va evolucionando a mejor en cada iteración del algoritmo (Lones, 2011). Dichas iteraciones se conocen como generaciones y, normalmente, la última generación incluye la mejor o las mejores soluciones a nuestro problema de optimización. Cada posible solución a nuestro problema se conoce como individuo, y cada individuo codifica las variables

independientes del problema de optimización. Estas variables representan los genes de la cadena cromosómica que representa a un individuo. Los algoritmos genéticos están basados en la Teoría Evolucionista de Charles Darwin (Darwin, 1859). Dicha teoría, explicado de forma muy simple, indica que los individuos que mejor se adaptan al medio son aquellos que tienen más probabilidades de dejar descendencia, y cuyos genes pasarán a las siguientes generaciones. La teoría de Darwin también describe que aquellas modificaciones genéticas que hacen que los individuos se adapten mejor al medio, tienen mayor probabilidad de perdurar en el tiempo. Estas ideas son las que utilizan los algoritmos genéticos para realizar una búsqueda estocástica guiada de forma eficiente. En los problemas de optimización numéricos, los individuos son potenciales soluciones al problema y la adaptación al medio se mide mediante la función que queremos optimizar, también llamada función objetivo, *fitness function* o función de evaluación[1]. Un individuo se adaptará bien al medio si produce un desempeño o *fitness*[2] alto, en caso de que se quiera maximizar la función objetivo, o si produce un desempeño bajo en caso de que estemos ante un problema de minimización. Ambos problemas son siempre duales[3], por lo que pasar de un problema de maximización a un problema de minimización es tan sencillo como multiplicar por –1 el resultado de la función objetivo.

En cada iteración del algoritmo, nuevos individuos (descendientes o, en inglés, *offsprings*) son creados mediante operaciones genéticas, dando lugar a nuevas poblaciones. Dichas operaciones genéticas, que se pueden resumir en tres bloques -selección, cruce y mutación- son el motor de búsqueda de un algoritmo genético. Cada vez que se crea un nuevo conjunto de individuos, se crea una nueva generación, y dicho proceso termina con la generación final, la cual debe incluir los mejores individuos encontrados a lo largo de las generaciones. Así, la Figura 1.1 representa el funcionamiento general de un algoritmo genético. Como se puede observar, se parte de una población inicial aleatoria y, a través de las operaciones genéticas, se van obteniendo nuevas generaciones hasta que se alcanza la población final. En este primer capítulo del libro, vamos a entrar en detalle en cada uno de los pasos y mecanismos que conforman un algoritmo genético; para ello, utilizaremos la librería de *Python deap*[4]. Esta librería nos facilita el diseño e

implementación de distintos algoritmos genéticos, ya que incluye muchas funciones de librería que desarrollan los principales componentes de un algoritmo genético.

Figura 1.1. Esquema del funcionamiento de un algoritmo genético.

¿Por qué recurrimos a la optimización metaheurística?

A lo largo de nuestra vida académica y profesional como ingenieros, con frecuencia nos encontramos con problemas de optimización de gran complejidad. A veces, esta radica en la gran cantidad de variables que hay que manejar; otras, en la complejidad de las ecuaciones que las gobiernan. A veces, incluso nos planteamos si la solución a nuestro problema existe. Pero, en general, solemos decir que estos problemas son *difíciles de resolver*. Pero ¿qué significa que un problema sea difícil de resolver? Aunque pueda parecer una pregunta trivial, es la primera que debemos formular cuando nos planteamos el uso de optimización metaheurística. Por supuesto, no es una pregunta fácil de responder.

Los métodos de optimización se pueden clasificar en dos grandes grupos bien diferenciados: métodos exactos y métodos metaheurísticos o aproximados.

La diferencia fundamental entre ellos está clara: los métodos exactos garantizan la obtención de la solución óptima, mientras que los metaheurísticos no. Llegados a este punto, uno podría preguntarse qué sentido tiene inclinarse por la segunda opción pudiendo utilizar un método exacto. Pues bien, la realidad es que no siempre podemos encontrar un método exacto que permita resolver nuestro problema. Es más: si lo hay, es muy posible que su aplicación no sea viable para un problema de cierta complejidad; por ejemplo, por el tiempo de resolución (una búsqueda

extensiva para un problema combinatorio de algunos cientos de variables puede tardar meses o años[5]), o por las simplificaciones que pueden requerir para su aplicación (por ejemplo puede ser necesario linealizar las restricciones del problema). Además, las estrategias de resolución analíticas, como los métodos basados en gradiente, pueden converger a óptimos locales y no alcanzar el óptimo global del problema.

Así, para decidir qué estrategia utilizar para abordar un problema *difícil de resolver* deberíamos plantearnos, al menos, las siguientes cuestiones:

- **¿Cómo de grande es mi problema?**
  Evaluar el tamaño del espacio de búsqueda (esto es, el número de soluciones posibles) es un buen indicador de la complejidad del problema. Si conocemos el tiempo necesario para evaluar una solución, podemos hacer una estimación del tiempo que sería necesario para realizar una búsqueda extensiva.
- **¿Necesito resolverlo rápido?**
  Está claro que es preferible disponer de la solución lo antes posible, pero debemos pensar si realmente necesitamos que nuestro problema se resuelva en cuestión de segundos o si, por el contrario, varias horas (o días) son un plazo aceptable.
- **¿Hay muchas restricciones? ¿Cómo son?**
  Un elevado número de restricciones, y, -sobretodo, una gran cantidad de no linealidades en las mismaspuede constituir un obstáculo insalvable para abordar analíticamente nuestro problema de optimización. Resolver de forma analítica una versión suficientemente simplificada (por ejemplo, relajando ciertas restricciones) de nuestro problema puede ser una buena idea, y nos puede ayudar en el desarrollo de estrategias metaheurísticas para el problema completo.
- **¿Qué precisión necesito en los resultados?**
  Cuanto más precisa sea nuestra solución mejor, por supuesto. Pero ¿cuánto es suficiente? ¿Considerarías adecuada una solución un 1% peor que el óptimo global? ¿Y un 5%? Es posible que con un 10% tu

solución sea más que útil para cumplir su propósito, y te permitiría ahorrar una gran parte de tiempo o de recursos.

Limitaciones de los métodos tradicionales basados en gradiente

Tradicionalmente, en los cursos de cálculo, tanto en enseñanza secundaria como en niveles superiores, los métodos de optimización estudiados son los métodos basados en gradiente. De forma genérica, para una función $f(x)$ el procedimiento consiste en:

1. Calculamos las derivadas de la función $f'(x)$.
2. Obtenemos los puntos para los que el gradiente se hace cero $f'(x_i) = 0$.
3. Calculamos la segunda deriva $f''(x)$ y evaluamos los puntos anteriores para saber si es un máximo $f''(x_i < 0)$ o un mínimo $f''(x_i > 0)$.

Cuando tenemos problemas con varias variables, debemos trabajar con derivadas parciales y obtener las matrices Hessianas. Como podemos observar, los métodos basados en gradientes se basan en el cálculo de las derivadas de la función (o derivadas parciales). Sin embargo, existen muchos casos en los que el cálculo de las derivadas es muy complejo o incluso imposible. Imaginemos que queremos optimizar el funcionamiento una planta industrial de la que no tenemos el modelo, pero sí tenemos un *software* de simulación de la planta. En esta situación, no podemos aplicar los métodos basados en gradiente, pero sí podremos aplicar los métodos metaheurísticos, como veremos más adelante. Otro problema de los métodos basados en gradiente es que se pueden quedar atrapados en óptimos locales, ya que pueden existir varios puntos de la función en los que el gradiente se haga cero. Por último, también es importante indicar que en los métodos numéricos basados en gradiente, se debe indicar un punto inicial. Por lo tanto, el funcionamiento del algoritmo dependerá de la selección de dicho punto, y pueden aparecer problemas de convergencia en algunos casos.

A continuación, veremos que los algoritmos metaheurísticos -y en concreto los algoritmos genéticos- nos permiten obtener soluciones realmente buenas a problemas en los que los métodos tradicionales basados en gradiente pueden presentar problemas.

1.2 Primeros pasos mediante un problema sencillo

Como la mejor forma de aprender a programar es simplemente programando, vamos a resolver un problema sencillo paso a paso, con el fin de poder describir los distintos componentes de un algoritmo genético, así como su implementación en *Python*.

Ejercicio 1.1 Se desea encontrar el máximo de la función

$$f(x,y) = \sqrt{x^2+y^2},$$

en el dominio $\{x, y\} \in [-100, 100]$.

En este simple ejemplo, las variables independientes son x e y, y la función objetivo o función de *fitness* es $f(x,y)$. Un individuo, pues, debe codificar dichas variables independientes en una cadena cromosómica (información genética del individuo) en la que cada variable independiente corresponde a un gen. Así, en nuestro problema ejemplo, la cadena cromosómica estaría formada por dos genes que al confinarse en forma de lista, quedarían como $[x_i, y_i]$, con $i = 1,...,n$ (siendo n el número de individuos que componen la población). La Figura 1.2 muestra gráficamente la representación de un individuo y de una población de n individuos.

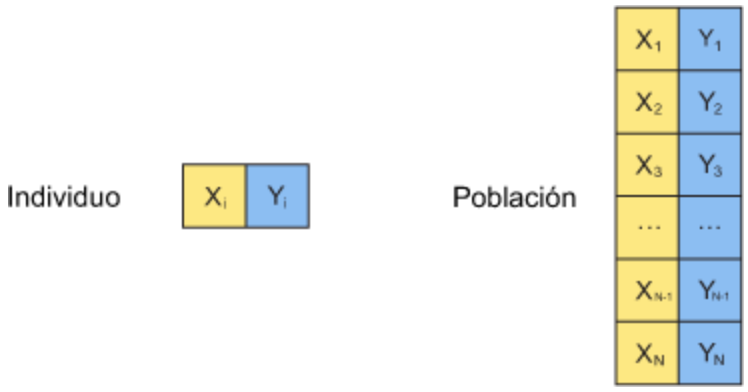


Figura 1.2. Representación de un individuo y de una población.

En principio, vamos a considerar que la población del algoritmo no cambia de tamaño a lo largo de las generaciones; por lo tanto, n será constante.

En los algoritmos genéticos tradicionales, el tamaño de la población es siempre constante. Además, un aspecto importante en los algoritmos genéticos es que la población inicial, en la mayoría de los casos, se elige de manera aleatoria con el fin de partir de una diversidad amplia de individuos.

Es decir, queremos tener genes de muchos tipos. En caso contrario, si los individuos de la población inicial se parecieran mucho, estaríamos limitando el proceso de búsqueda del algoritmo genético. Por lo tanto, a la hora de abordar la resolución de un problema mediante algoritmos genéticos, uno de los primeros pasos que tenemos que dar es buscar un mecanismo para generar soluciones aleatorias a nuestro problema que difieran lo suficiente las unas de las otras. Imaginemos que la población inicial está compuesta por diez individuos (n = 10); en consecuencia, se deberán generar diez soluciones aleatorias. En la Tabla 1.1 se muestran las soluciones iniciales generadas, siendo esta solo una posible muestra de diez soluciones aleatorias.

Tabla 1.1. Población inicial.

Individuo	x	y
1	68.88	81.62
2	51.59	0.93
3	-15.88	-43.63