¿Qué es un algoritmo?

En este notebook, empezaremos nuestro estudio de algoritmos computacionales.

Un **algoritmo** es una "receta" computacional, que consiste en una serie de instrucciones para que la computadora lleve a cabo un cálculo dado. Gran parte del curso consistirá en desarrollar algoritmos para calcular, de forma numérica, distintas cantidades en la física, a partir de algún modelo matemático. El campo que se ocupa de diseñar y estudiar estos algoritmos es el **análisis numérico**. Su aplicación a problemas de física se puede decir que constituye la **física computacional**.

Algunos algoritmos (por ejemplo, la eliminación gaussiana que veremos más adelante) proveen una manera de llevar a cabo un cálculo de manera "exacta" (dentro de las restricciones impuestas por el uso de números con precisión finita) en un número finito de pasos.

Sin embargo, en general, no podemos esperar que haya una fórmula analítica cerrada para calcular las cantidades de interés de manera exacta. En este caso, será necesario emplear un algoritmo **iterativo**, que en principio podría correr ¡por un tiempo infinito! Lo pararemos cuando pensemos que el problema ya se resolvió de forma "suficientemente buena".

Algoritmos iterativos

Un **algoritmo iterativo** repite un mismo cálculo un gran número de veces, modificando un valor (o varios valores) en el proceso, hasta que (en el mejor de los casos) converja a una solución.

Un algorithmo iterativo empieza desde una adivinanza inicial x_0, y aplica un procedimiento / receta matemática, o sea alguna función f (que puede ser complicada), para producir una siguiente adivinanza $x_1 := f(x_0)$. Esto se repite para producir una secuencia $x_0, x_1, \ldots, x_n, \ldots$, tales que

$$x_1 = f(x_0)$$

$$x_2 = f(x_1)$$

$$x_3 = f(x_2)$$

etc. En general, escribimos una iteración como

$$x_{n+1} := f(x_n).$$

La esperanza es que la secuencia x_n converja hacia un valor límite $x^{cuando} n \to \infty , y queel x^$ resultante sea solución del problema original.

Dado que no podemos llevar a cabo la iteración un número infinito de veces, se corta la iteración después de un cierto número de pasos, para dar una solución *aproximada*, que se acerca dentro de cierta *tolerancia* al resultado teórico exacto x^*. Por lo tanto, cualquier algoritmo iterativo requiere una condición de terminación.

En la computadora, no pensamos en escribir x_n, sino pensamos en el **valor actual** de x, y el **valor siguiente** de x. Dentro del bucle, usamos el valor actual de x para calcular el valor nuevo. Al final del bucle, debemos actualizar el "valor actual".

Iteraciones de punto fijo

[1] (i) Define la función $f_1(x) = \frac{x}{2} - 1$.

(ii) Toma una condición inicial x_0 y lleva a cabo la iteración a mano. ¿Qué observas?

(iii) Utiliza un bucle `for` para ver cómo son los primeros N iterados x_n. Haz una función que tome como argumento x_0 y N.

(iv) Repite la iteración para varios valores de x_0. ¿Qué observas?

[2] (i) Define una función `iterar` que lleva a cabo lo que hiciste en la pregunta 1. Debe aceptar como su primer argumento (el nombre de) *la función `f` que iterar*, así como el número de veces que se iterará, y la condición inicial.

(ii) Utilízalo para iterar la función $f_2(x) = \cos(x)$.

(iii) ¿Adónde converge la iteración? ¿Cuál ecuación hemos resuelto?

[3] Ahora queremos graficar.

(i) Modifica la función `iterar` para que vaya guardando los valores de x_n en un arreglo, el cual regresa.

(ii) Utiliza el paquete `Plots` para graficar el resultado. ¡Ten cuidado con el tipo de gráfica que dibujas: deben ser puntos! Para eso puedes utilizar la función `scatter`.

(iii) Grafica la trayectoria para varios valores de x_0 en una sola gráfica. [Utiliza `scatter!`, con `!` al final, para *agregar* más información a un `plot` ya existente.] ¿Qué observas?

(iv) Importa el paquete `Interact` y utiliza `@manipulate` antes de un bucle `for` sobre `x_0` para ver cómo cambia la visualización (de una sola condición inicial) de forma interactiva.

[4] ¿Qué ocurre si iteras la función $f_3(x) = 2x + 1$?

[5] (i) Pensando en la ecuación $x_{n+1} = f(x_n)$, en el límite cuando $n \to \infty$, si es que la iteración converge a un valor que podemos llamar x_∞, ¿a cuál valor debe converger? [Pista: toma el límite de los dos lados de la ecuación.] ¿Coincide con lo observado para f_1 y f_2?

(ii) Graficamente, ¿a qué corresponde resolver la ecuación para x_∞? Dibuja las gráficas correspondientes para f_1 y f_2 y checa tu respuesta.

(iii) Puedes adivinar cuál es la condición para que la iteración converja?

[6] A menudo, se puede utilizar una iteración de este tipo para resolver ecuaciones.

(i) Inventa una iteración de la forma $x_{n+1} = f(x_n)$ para resolver la ecuación $x^2 + x - 1 = 0$. ¿Para cuáles x_0 funciona? ¿A cuál solución converge?

(ii) Inventa otra. ¿Funciona para x_0 diferentes?

(iii) Nota que hay algunas iteraciones que **no converjan**. Por ejemplo, ¿qué ocurre con la iteración $x_{n+1} = 1 - x_n^2$?

Bucles `while`

En lo anterior, usamos un bucle `for`, que requiere que sepamos de antemano el número de iteraciones que queramos. Sin embargo, en este tipo de problemas, es más natural esperar **hasta que** converja, sin saber cuánto tiempo tomará.

Para esto, podemos ocupar otro tipo de bucle, un bucle `while` ("mientras", en español). Un bucle de este tipo repite los comandos en el cuerpo del bucle **mientras** una condición siga cierta. Su sintaxis es como sigue:

```
while <condicion>
    [haz esto]
    [y esto]
end
```

Sin embargo, para evitar bucles infinitos, a menudo es sensato incluir un contador para que no pueda haber demasiadas (posiblemente infinitas) iteraciones.

Mientras que en un bucle `for` hay un contador que se actualiza automáticamente, en un bucle `while` **nosotros somos los responsables** de tener una variable que actúe como contador.

[7] (i) Utilice un bucle `while` para contar de 1 a 10.

(ii) Utilice un bucle `while` para encontrar la potencia de 2 más grande debajo de un número dado.

(iii) Repite lo mismo con un bucle `for`, usando la palabra clave `break` para salir del bucle cuando una cierta condición se satisfaga.

Bucles while e iteraciones de punto fijo

[8] Utiliza un bucle `while` para la iteración de la función f_1. Fija una **tolerancia** razonable, y repite la iteración **hasta que** la distancia entre un iterado y el siguiente sea menor a la tolerancia. [Pista: ¿Cuál función matemática da la distancia entre dos números en una dimensión. Encuentra la función de Julia que lo hace.]

[9] De la misma forma, escribe una versión nueva de la función `iterar`, que utiliza un `while` y acepta una tolerancia como argumento.

Haz tu propia biblioteca

Guarda la función `iterar` en un archivo que se llama `herramientas.jl`. Iremos agregando más métodos a este archivo.

Se incluye en un notebook o un script de Julia con `include("herramientas.jl")`.