Ejercicios Curso C++

Victor Buendía Ruiz-Azuaga

Estudiante de Doctorado.
Universidad de Granada, Departamento de Electromagnetismo y Física de la Materia
Università di Parma, Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Ninguno de estos ejercicios es obligatorio. Si los trabajáis y tenéis cualquier duda con ellos contactad conmigo por correo.

1 Repaso de C++

1. **Ordenar tres números:** Dados tres números introducidos por el usuario, mostrarlos por pantalla de mayor a menor.

2. **La serie geométrica:** es posible demostrar que la serie geométrica viene dada por:

$$g\left(x,N\right)=\sum_{n=0}^{N-1}x^{n}=\frac{1-x^{N}}{1-x}.$$

   Crea una función que calcule y represente $g\left(x\right)$ en el intervalo $x\in\left[0,1\right]$ usando la expresión con la sumatoria. Compárala con el miembro de la derecha. ¿Qué ocurre cuando $N\rightarrow+\infty$?

3. **La función de Weiertrass:** aunque parezca increíble, hay funciones que son continuas en todo $\mathbb{R}$, ¡pero no son derivables en *ningún* punto! El primer ejemplo de este hecho lo dio Weiertrass en 1872 con la función:

$$W\left(x\right)=\sum_{n=0}^{+\infty}a^{n}\cos\left(b^{n}\pi x\right)$$

   Crea una función que calcule $W\left(x\right)$, y represéntala para los valores $a=1$, $b=7$ para varios valores de x, por ejemplo $x\in\left[-1,1\right]$, $x\in\left[-0.1,0.1\right]$ y $x\in\left[-0.001,0.001\right]$. ¿Dejan de aparecer picos en algún punto?
   Extra: los parámetros deben cumplir la condición $ab>1+3\pi/2$, con b impar, para conseguir el efecto deseado. Añade a la función una condición para que advierta al usuario si esta condición no se cumple.

4. **Calculando binomiales:** los coeficientes binomiales pueden calcularse de forma eficiente a partir de la siguiente expresión:

$$\binom{n}{k}=\frac{n!}{k!\left(n-k\right)!}=\prod_{j=1}^{k}\frac{n+1-j}{j}$$

comprueba que para coeficientes binomiales muy grandes, como

$$\binom{200}{50} = 1.00891 \cdot 10^{29}$$

la definición directa con los factoriales es incalculable para el ordenador (ya que $200! \sim 10^{374}$, que no puede representarse en doble precisión), mientras que la segunda es estable numéricamente. Comprueba, además, que la definición directa es más lenta, ya que involucra tres bucles con un total de $2n$ iteraciones, mientras que la segunda solo involucra k iteraciones.
Extra: comprueba que en todo caso, para valores de n y k muy grandes, podemos calcular analíticamente $\log \binom{n}{k}$ con la aproximación de Stirling $\log(n!) \simeq n \log n - n$ permite calcular directamente el valor sin requerir una sumatoria (¡y por tanto, es mucho más eficiente!).

5. **Búsqueda binaria:** a menudo es necesario realizar una búsqueda en un array *ordenado*, y devolver la posición en el array del elemento que buscamos. Un procedimiento muy utilizado por su sencillez y eficiencia es la búsqueda binaria. Implementa la búsqueda binaria en una función. El algoritmo tiene los siguientes pasos (se pueden encontrar pseudocódigos fácilmente por internet):

   (a) Tenemos un array con N elementos, y un elemento x que queremos encontrar. Cogemos el elemento $N/2$ del array, digamos que se llama y. Entonces...

      i. Si $y > x$, entonces todos los elementos a la derecha de y también son mayores que x. Basta con quedarnos con el array de 0 a $N/2$ y comenzar de nuevo.

      ii. Si $y < x$, entonces todos los elementos a la izquierda de y también son menores que x. Basta con quedarnos con el array de de $N/2$ a N y comenzar de nuevo.

   (b) El proceso se detiene cuando $y = x$, o bien cuando el array final tiene solo dos elementos, y_1 e y_2, que cumplen $y_1 < x < y_2$. En este punto, devolvemos el índice correspondiente al elemento y, o bien avisamos al usuario de que la búsqueda no tuvo éxito.

6. **Traspuesta de una matriz:** crea una función que reciba como *único* argumento una matriz arbitraria $n \times m$. Al salir de la función, esa matriz estará traspuesta.

2 Clases

Esta sección incluye ideas de clases que se pueden realizar para practicar y pueden tener alguna utilidad a largo plazo. Dependiendo de las aplicaciones concretas de cada uno, pueden surgir más ideas.

1. **Gráficas en C++**: escribe una clase que utilice el comando *system* como base para integrar tu software favorito de gráficas en C++ (gnuplot, matplotlib...). Para ello, tu clase debe estar compuesta fundamentalmente por un *ofstream*. Después,

   (a) El constructor: el constructor iniciar el ofstream y abrir un fichero de script (digamos *grafica.plt*) en el que irán todos los comandos que necesitemos.

   (b) Funciones: necesitaremos funciones que permitan personalizar nuestra gráfica. Por ejemplo, una función *plot* que reciba como argumento la ruta de un fichero con datos. Esta función escribirá una línea en el ofstream que muestre la gráfica. Harán falta más líneas para las opciones de texto en eje X, eje Y, título, etc. Todo esto se va escribiendo en el fichero de script a base de funciones.

(c) Mostrar la gráfica: finalmente, usaremos system para mostrar el fichero que hemos ido construyendo (ejecutando por ejemplo *gnuplot grafica.plt*). Básicamente, esta interfaz permite escribir ficheros de gnuplot y representarlos sin necesidad de salir de C++.

2. **Ampliación de *polinomio:*** este ejercicio propone ampliar algunas funciones de la clase polinomio, sobrecargando nuevos operadores de utilidad y añadiendo funciones:

   (a) Sobrecargar el operador *()* de forma que al llamar a *p(double x0)* el usuario reciba el valor del polinomio evaluado en el punto x_0, es decir $p(x_0)$.

   (b) Hacer una función *derivada* que devuelva el polinomio resultante de derivar el primero. Realizar también la función *integral* que devuelva la integral del polinomio. La constante que viene de la integral será siempre 0.

   (c) Realizar una función *calcular_raices* que devuelva los valores de las raíces reales del polinomio. Para resolver la ecuación, puede aplicarse directamente el teorema de Bolzano: si $p(x_1) < 0$ y $p(x_2) > 0$, entonces hay una raíz x_0 tal que $x_1 < x_0 < x_2$. En este punto podemos aplicar el mismo razonamiento que en una búsqueda binaria (ver ejercicio más arriba). La función recibirá como argumento el valor inicial a partir del cual empezar a calcular.

   (d) Emplear las funciones creadas para hacer funciones *encuentra_minimo* y *encuentra_maximo* para encontrar extremos en un polinomio.

3 Biblioteca estándar

1. **Puliendo polinomio:** cambia los arrays estáticos de la biblioteca polinomio por arrays dinámicos.

2. **Operadores para vectores.** La clase *vector<tipo>* tiene muchas utilidades, pero todavía es posible mejorarla para nuestros propósitos. Por ejemplo, es posible hacer un fichero con utilidades extra para vectores, que incorpore por ejemplo sobrecargas de los operadores. Querríamos, por tanto:

   (a) Calcular media y desviación estándar de unos datos.

   (b) Poder sumar, restar, multiplicar y dividir vectores (elemento a elemento) mediante sobrecargas de operador.

   (c) Realizar comparaciones elemento a elemento del vector.

   (d) Poder mostrar un vector por pantalla con << o leerlo desde fichero con un único >>.

3. **El fractal de Mandelbrot:** el conjunto de Mandelbrot es un subconjunto del plano complejo $\mathbb{C}$ formado por todos los números w tales que

$$z_{n+1} = z_n^2 + w$$

   es convergente. siendo $z_0 = 0$. Calcula y representa el conjunto de Mandelbrot, sabiendo que los puntos con $|w|^2 > 2$ no pertenecen al conjunto.
   Extra: a la hora de representarlo, se puede hacer con colores, poniendo un valor 0 a los puntos dentro del conjunto (es decir, aquellos con una sucesión convergente) y un valor positivo a los que no pertenecen al conjunto. Una forma de clasificar los puntos fuera del conjunto es asignarle el número de iteraciones necesarias para asegurarse de que un punto no pertenece al conjunto, es decir, el $n+1$ tal que$|z_{n+1}|^2 > 2$.

4. **El fractal de Julia:** en realidad, el conjunto de Mandelbrot es un caso especial del conjunto de Julia. Este permite encontrar formas mucho más variadas. Básicamente, dado un punto w del plano complejo, un conjunto de Julia tiene la forma

$$z_{n+1} = f(z_n)$$

con $z_0 = w$. Aquí tienes algunos ejemplos de conjuntos de Julia, obtenidos de la Wikipedia. Una vez el código está hecho, puedes probar con las combinaciones que se te ocurran:

   (a) El círculo unidad:
   $$f(z) = z^2$$

   (b) Un mapa cuadrático:
   $$f(z) = z^2 + 0.285 - 0.01i$$

   (c) Un mapa exponencial:
   $$f(z) = \exp\left(z^3\right) - 0.59$$

   (d) Algo más complicado:
   $$f(z) = \sqrt{\sinh\left(z^2\right)} + 0.065 + 0.122i$$

5. **El número π en una diana:** sea un círculo de radio $a = 1$ inscrito en un cuadrado de lado $2a$. Supongamos que lanzamos un punto al azar en el cuadrado. La probabilidad de que caiga dentro del círculo es:

$$p = \frac{\pi a^2}{(2a)^2} = \frac{\pi}{4}$$

Podemos obtener π (de una forma muy ineficiente) si estimamos la probabilidad p numéricamente. Para hacerlo, lanza 10^5 puntos aleatorios en el cuadrado, y obtén:

$$p = \frac{\#\text{puntos dentro del circulo}}{\#\text{puntos lanzados} = 10^5}$$

6. **Integrales Monte Carlo:** el método anterior es lo que se puede considerar un método de Monte Carlo para resolver integrales numéricamente. Su eficiencia es inigualable para resolver integrales en muchas dimensiones. Vamos a realizar dos ejemplos sencillos de integrales Monte Carlo:

   (a) *Aceptación o Rechazo:* comprobar que

   $$\int_0^1 dx \sin\left(1 - x^2\right) = 0.593492$$

   mediante el método del ejercicio anterior: lanzando puntos aleatorios en el cuadrado $x \in [0, 1]$, $y \in [0, 1]$ y calculando la probabilidad p de queden dentro de la región determinada por $\sin\left(1 - x^2\right)$. Esta probabilidad nos dará como resultado el valor de la integral.

   (b) *Trucos de estadística.* Vamos a tomar la siguiente integral:

   $$\int_0^{+\infty} x^2 e^{-x} dx = 2.$$

Si sabemos algo de estadística, podemos recordar que dada una función densidad de probabilidad $f(x)$, el segundo momento de la distribución es la media de los valores al cuadrado:

$$E\left(x^2\right) = \int_{\mathbb{R}} x^2 f\left(x^2\right)$$

Observa que nuestra integral simplemente el segundo momento de la distribución exponencial:

$$f(x) = \begin{cases} 0 & x < 0 \\ e^{-x} & x \geq 0 \end{cases}$$

Por tanto, es posible evaluar la integral calculando el momento de la distribución. Para ello:

i. Genera muchos números obtenidos a partir de una distribución exponencial, usando la STL de C++.
ii. Calcula la media del cuadrado de los números.