Algoritmica
Practica 2. Serie Unimodal
2018-2019

UNIVERSIDAD
DE GRANADA

Félix Ramírez García
Raúl Del Pozo Moreno
Cristian Piles Ruiz
Oleksandr Kudryavtsev
Sixto Coca Cruz

INDICE

1. Complejidad y explicación del algoritmo de serie unimodal 3
2. Comparación de los dos algoritmos (DyV y 'obvio') 6
3. Eficiencia del algoritmo "divide y vencerás" 7
   3.1. Eficiencia empirica 7
   3.2. Eficiencia hibrida 9

1. Complejidad y explicación del algoritmo de serie unimodal

Para empezar con esta práctica vamos a exponer el problema que hay que resolver, para ello hemos implementado una solución ‘obvia’ y otra utilizando la técnica de divide y vencerás. El problema es el siguiente:

Sea un vector v de números de tamaño n, todos distintos, de forma que existe un índice p (que no es ni el primero ni el último) tal que a la izquierda de p los números están ordenados de forma creciente y a la derecha de p están ordenados de forma decreciente; es decir :

$$\forall i, j \leq p, i < j \Rightarrow v[i] < v[j] \text{ y } \forall i, j \geq p, i < j \Rightarrow v[i] > v[j]$$

La solución ‘obvia’ que hemos usado es recorrer la parte creciente del vector hasta encontrar el primer elemento decreciente, siendo p el elemento anterior a éste.
El código empleado para este análisis ‘obvio’ es el siguiente:

```
int buscar(int *T, int n){
    int pivote = -1;
    int anterior = T[0];

    for(int i = 0 ; i < n && pivote == -1 ; ++i){
        if(anterior > T[i]){
            pivote = anterior;
        }
        anterior = T[i];
    }

    return pivote;
}
```

Este código tienen una eficiencia de O(n) , que se comparará gráficamente con la de la solución de divide y vencerás.

La solución de este problema con la técnica de divide y vencerás ha sido implementar el siguiente algoritmo recursivo:

```
int buscarDyV(const std::vector<int> &v, int inicio, int fin) {
    int tamanio = fin - inicio + 1;

    if(tamanio > 2) {
        int mitad = (tamanio/2) + inicio;
        int izquierda = v[mitad-1];
        int derecha = v[mitad+1];
        int valor_mitad = v[mitad];

        //caso base 1, valor central es el mayor
        if(valor_mitad>izquierda && valor_mitad>derecha)
            return valor_mitad;
        else {
            //coger la primera mitad desde valor_mitad
            if(valor_mitad > derecha) {
                return buscarDyV(v,inicio,mitad);
            }
            //coger la segunda mitad desde valor_mitad
            if(valor_mitad < derecha) {
                return buscarDyV(v,mitad+1,fin);
            }
        }
    }
    //caso base 2, vector con n <= 2
    else {
        //cuando quedan dos, coje el mayor
        if(tamanio==2) {
            if(v[inicio]>v[fin])
                return v[inicio];
            else
                return v[fin];
        }
        //si queda uno, lo devuelve
        else
            return v[inicio];
    }
}
```

Para la implementación de este algoritmo se han tenido en cuenta los dos casos base en los que se obtiene la solución directa al problema. El primer caso base se da cuando el vector tiene un tamaño superior a 2 y el valor del elemento central del vector sea mayor que el valor del elemento de su izquierda y sea mayor que el elemento de su derecha. El segundo caso base se da cuando el tamaño del vector es menor o igual a dos, devolviendo el elemento mayor en el caso de que el tamaño sea dos o devolviendo el único elemento en el caso de que el tamaño del vector sea 1.

Partiendo de que no suceda ningún caso base el problema se divide en otro subproblema de tamaño n/2 de la siguiente forma:

- Si el valor del elemento de la mitad es mayor que el elemento de su derecha se vuelve a llamar a la función con la mitad del vector, concretamente con la mitad izquierda.
- Si el valor del elemento de la mitad es mayor que el elemento de su izquierda se vuelve a llamar a la función con la otra mitad del vector , la de la derecha.

El siguiente ejemplo gráfico muestra la búsqueda del mayor elemento en el siguiente vector de 10 elementos:

0	9	8	7	6	5	4	3	2	1

En la primera iteración de la solución divide y vencerás almacenamos las siguientes variables:

Mitad = 5 (Posicion central del vector)
Izquierda = 6 (Valor del elemento de la izquierda respecto a la posición central)
Derecha = 4 (Valor del elemento de la derecha respecto a la posición central)
Valor_mitad = 5 (Valor del elemento central)

Por lo que se reduciría el problema en la siguiente iteración a la mitad izquierda del vector anterior, siendo la siguiente:

0	9	8	7	6

En la segunda iteración almacenamos las siguientes variables:

Mitad = 2 (Posicion central del vector)
Izquierda = 9 (Valor del elemento de la izquierda respecto a la posición central)
Derecha = 7 (Valor del elemento de la derecha respecto a la posición central)
Valor_mitad = 8 (Valor del elemento central)

Por lo que se reduciría el problema en la siguiente iteración a la mitad izquierda del vector anterior, siendo la siguiente:

0	9	8

En la tercera iteración almacenamos las siguientes variables:

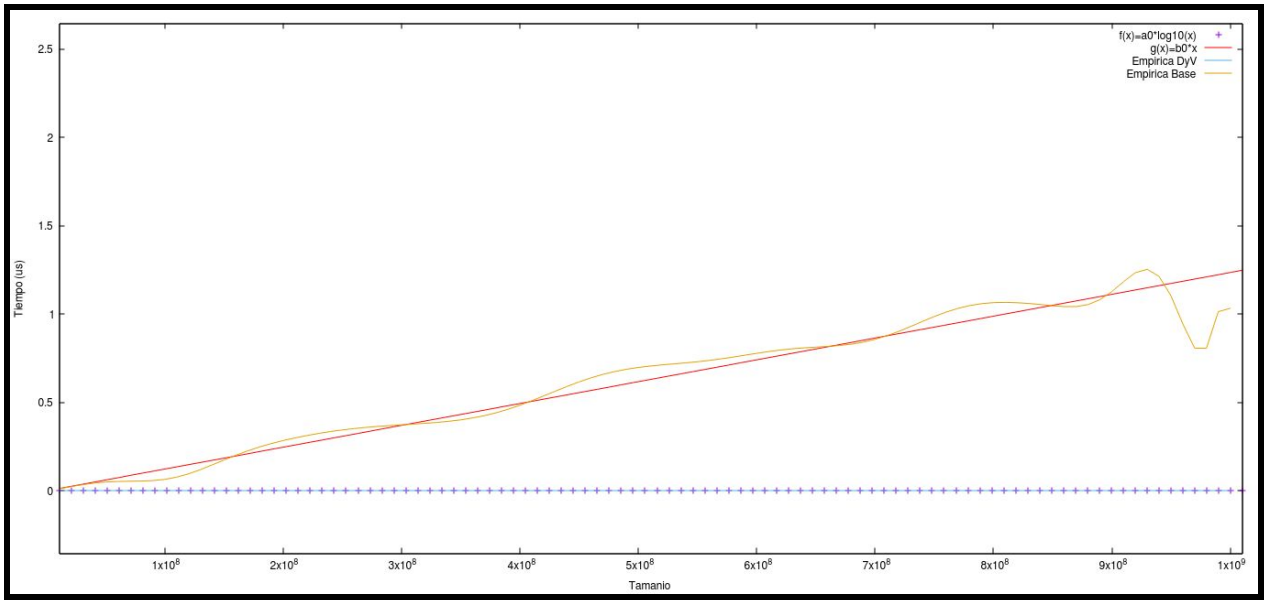
Mitad = 1 (Posicion central del vector)
Izquierda = 0 (Valor del elemento de la izquierda respecto a la posición central)
Derecha = 8 (Valor del elemento de la derecha respecto a la posición central)
Valor_mitad = 9 (Valor del elemento central)

En esta iteración ya se cumple el primer caso base , en el que la que el valor de la posición central es mayor que el valor de la posición izquierda y mayor que el valor de la posición de la derecha, por lo que el valor mayor del vector es el de la posición central.

2. Comparación de los dos algoritmos

Para la comparación de ambos algoritmos se han representado junto a sus funciones teóricas, como se puede ver en la imagen siguiente, existe una línea de color rojo de forma ascendente, la cual corresponde con la función teórica del algoritmo base, con una eficiencia O(n).

Junto a ella se puede apreciar otra línea de color amarillo que aunque no es estrictamente recta, casi se ajusta a la línea teórica, lo que indica que la eficiencia híbrida es también lineal.

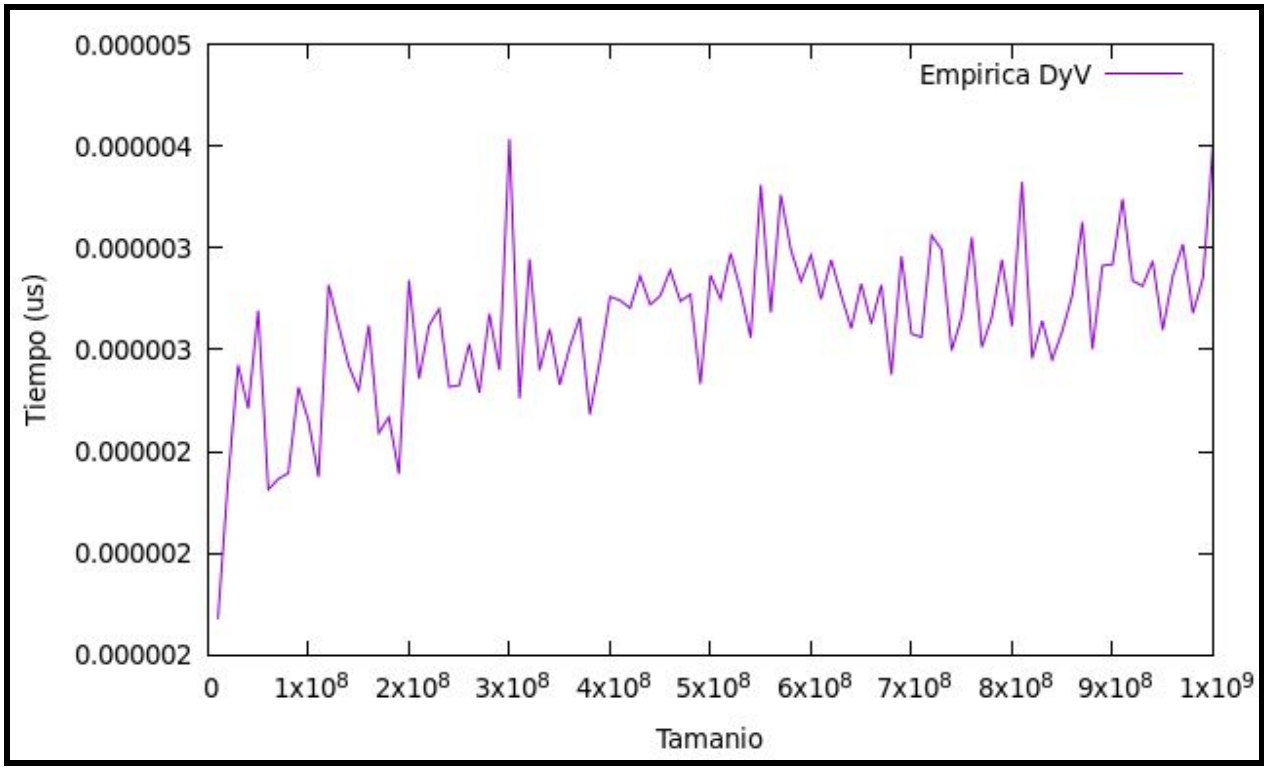


En la parte inferior, pegado al eje horizontal, se puede observar una línea azul y una serie de puntos (+), los puntos representan la función teórica correspondiente a un logaritmo, y la línea azul representa la eficiencia híbrida del algoritmo base mejorado con DyV. Como se puede ver, ambas funciones se ajustan muy bien, lo que corrobora que el algoritmo DyV que se ha implementado tiene una eficiencia O(log(x)).

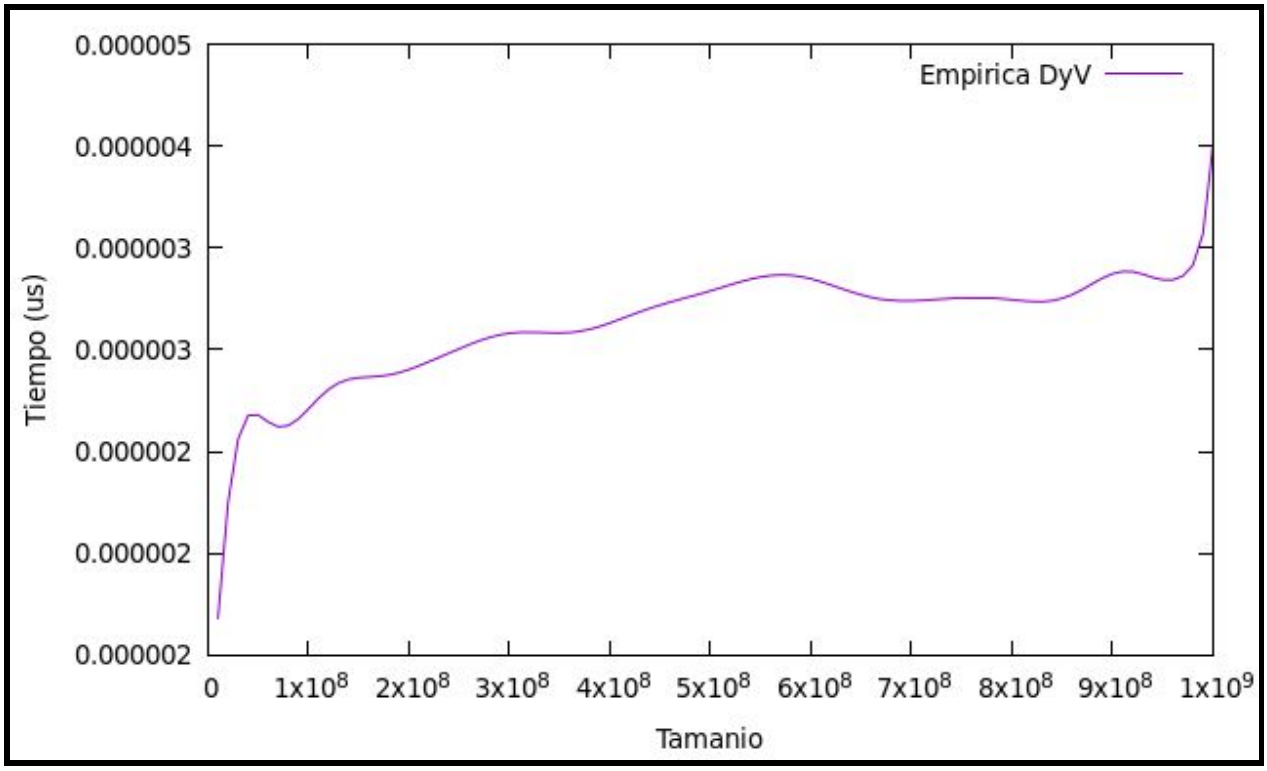
3. Eficiencia del algoritmo “divide y vencerás”

3.1 Eficiencia empirica

En la siguiente imagen se puede observar la eficiencia obtenida a partir de los datos de tiempos al ejecutar el algoritmo. Aunque se pueden observar muchos picos, se puede ver una curva ascendente correspondiente a un logaritmo.

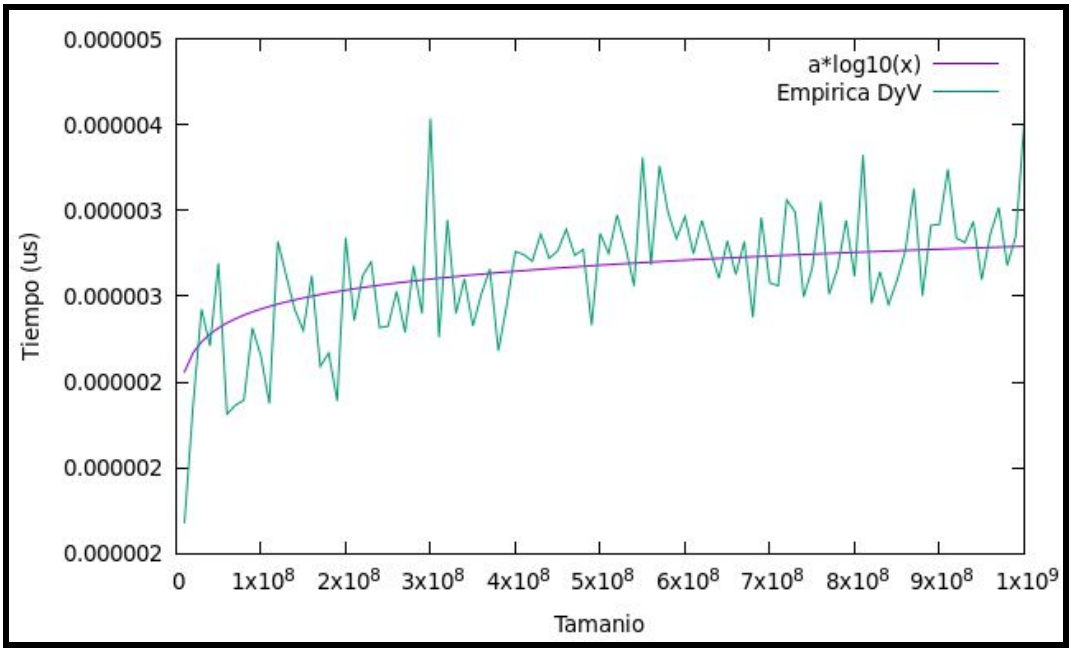


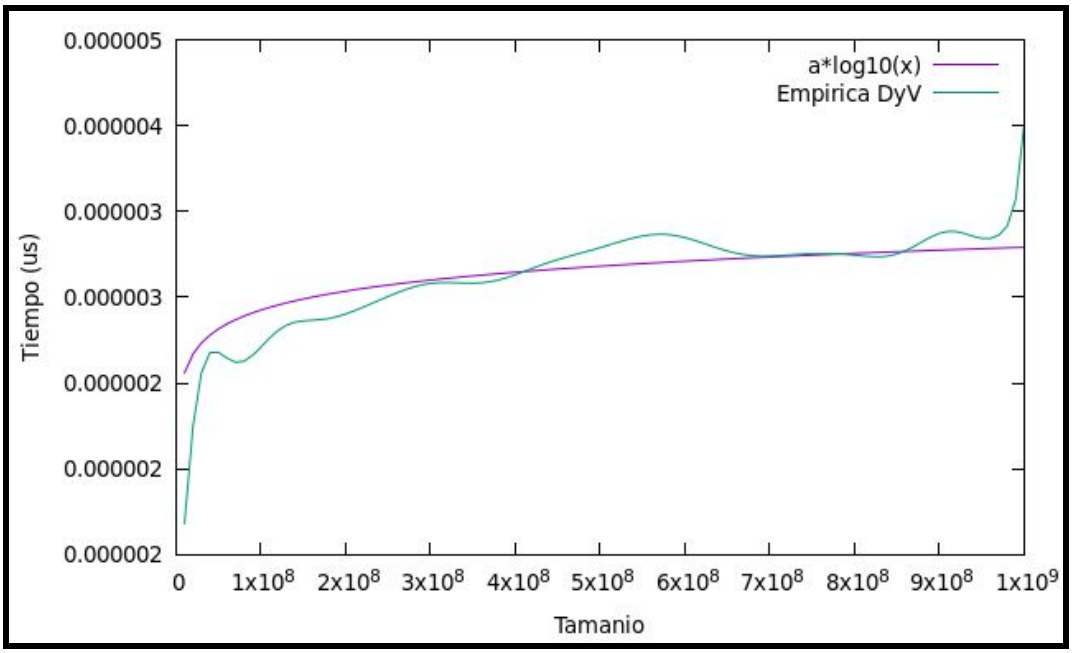
Esto se puede observar mejor si la gráfica se obtiene con la opción “smooth bezier”. De esta forma si se observa mejor la curva característica de un logaritmo, con la excepción de un dato final que produce un pico ascendente, posiblemente producido por algún ruido).


Empirica DyV
Tiempo (us)
Tamanio
0.000005
0.000004
0.000003
0.000003
0.000002
0.000002
0.000002
0
1x10^8
2x10^8
3x10^8
4x10^8
5x10^8
6x10^8
7x10^8
8x10^8
9x10^8
1x10^9

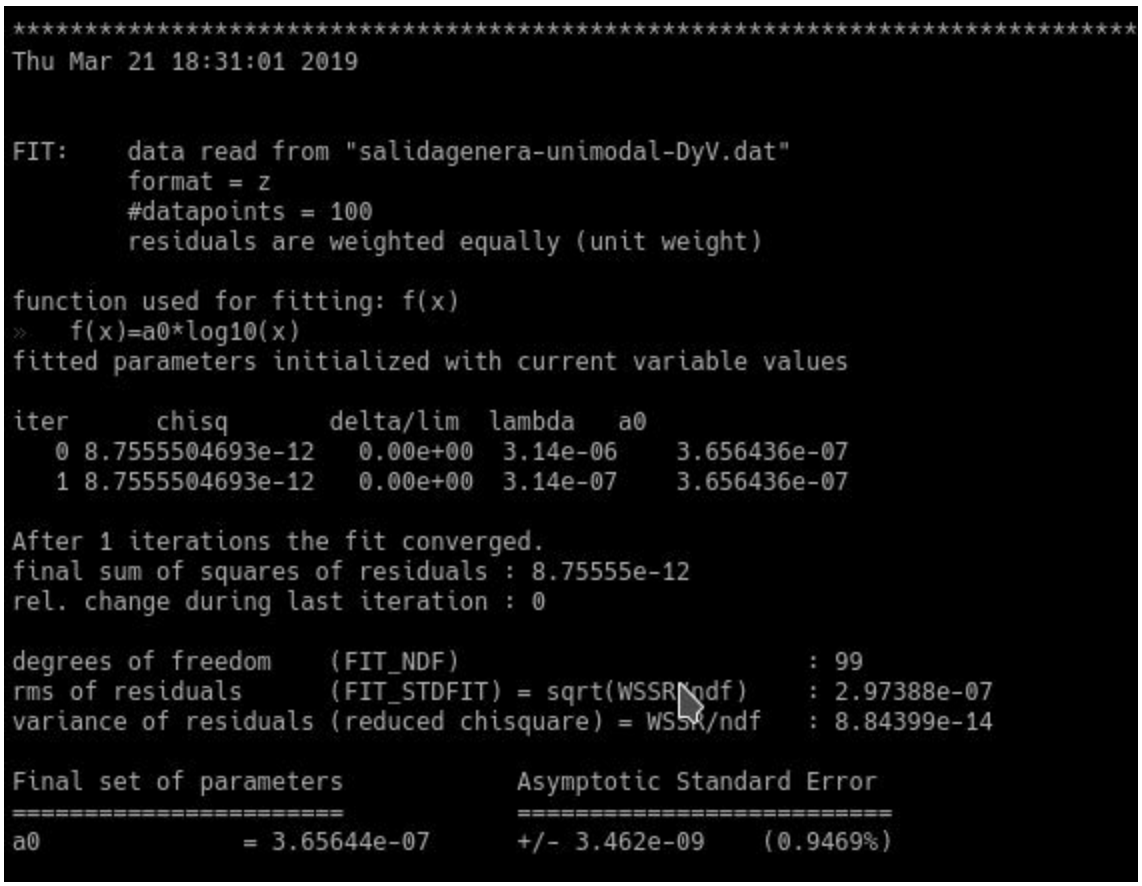
3.2 Eficiencia hibrida

Para la eficiencia híbrida se han comparados los datos obtenido de la empírica, con la curva ajustada de un logaritmo. Como se puede ver en las dos siguientes imágenes, las curvas se ajustan muy bien, cosa que se aprecia mejor con un suavizado.





A continuación se muestra el ajuste realizado entre la función logarítmica y los datos empíricos:



```
*******************************************************************************
Thu Mar 21 18:31:01 2019


FIT:    data read from "salidagenera-unimodal-DyV.dat"
        format = z
        #datapoints = 100
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
»   f(x)=a0*log10(x)
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a0
   0 8.7555504693e-12   0.00e+00  3.14e-06    3.656436e-07
   1 8.7555504693e-12   0.00e+00  3.14e-07    3.656436e-07

After 1 iterations the fit converged.
final sum of squares of residuals : 8.75555e-12
rel. change during last iteration : 0

degrees of freedom    (FIT_NDF)                        : 99
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 2.97388e-07
variance of residuals (reduced chisquare) = WSSR/ndf   : 8.84399e-14

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 3.65644e-07      +/- 3.462e-09    (0.9469%)
```

Como se puede ver, se ha obtenido un error del 0’9469%, un error aceptable.