PHÁT TRIỂN CHƯƠNG TRÌNH CON LÀM KHỚP DỮ LIỆU VỚI NHIỀU MÔ HÌNH

***ThS. Nguyễn Ngọc Anh*[1]**
***ThS. Trương Văn Minh*[2]**

TÓM TẮT

Hiện nay, có rất nhiều phần mềm máy tính cho phép người dùng làm khớp dữ liệu thực nghiệm với dạng hàm tùy ý nhập bởi người dùng. Tuy nhiên, các chương trình này có dạng đóng (đối với các chương trình thương mại) hoặc có hệ thống thư viện liên kết rất phức tạp (đối với các chương trình mã nguồn mở). Do đó, việc tận dụng thư viện của các chương trình này để nhúng vào các chương trình phần mềm nhỏ tự thiết kế là không thích hợp. Bài báo này đưa ra bộ chương trình con, cho phép người dùng làm khớp số liệu thực nghiệm với dạng hàm tùy ý, được viết bằng ngôn ngữ C++, có cấu trúc đơn giản, gói gọn trong một tập tin chỉ dài 438 dòng, thuận tiện để nhúng vào các chương trình tự phát triển. Kết quả thu được bằng chương trình được so sánh với ROOT.

Từ khóa:*Chương trình làm khớp nền C++, thuật toán làm khớp Levenberg–Marquardt*

1. Giới thiệu

Làm khớp dữ liệu theo một mô hình (dạng hàm) là một thủ tục được tiến hành rất phổ biến trong phân tích số liệu (phân tích phổ, xây dựng mô hình, xác định các tham số để nội suy, ngoại suy). Các thủ tục này có thể được thực hiện bởi các chương trình có giao diện trực quan như Origin [1], SciDavis [2] hoặc các chương trình dưới dạng lệnh thực thi như ROOT [3], R [4], Matlab [5], Gnuplot [6]. Tuy nhiên, một số là các chương trình thương mại (Origin, Matlab), do đó người sử dụng sẽ phải bỏ ra một chi phí không nhỏ để trang bị phần mềm. Tiếp nữa, các chương trình này thường có bộ thư viện đi kèm rất lớn, và liên kết với nhau rất phức tạp. Do, đó việc nhúng các thư viện này vào các chương trình nhỏ tự viết là rất phức tạp, và làm tăng kích thước của chương trình.

Trong thực tế, tùy thuộc vào tình huống cụ thể, việc sử dụng các phần mềm lớn kể trên để làm khớp không phải lúc nào cũng thuận lợi: chương trình quá nặng; hệ điều hành không hỗ trợ; … Khi đó các phần mềm tự viết sẽ là một giải pháp thích hợp.

Bộ chương trình con được cung cấp trong bài báo này cho phép người dùng nhúng vào trong các phần mềm tự viết, để thực thi tác vụ làm khớp số liệu theo mô hình bất kỳ do người dùng khai báo, sử dụng thuật toán LEVENBERG-MARQUARDT [7]. Chương trình cho phép người dùng lựa chọn làm khớp có trọng số hoặc không có trọng số. Bộ chương trình con này có kích thước rất nhỏ, chỉ ~12 kb, gói gọn trong một tập tin *.h, thuận tiện để người dùng khai báo trong chương trình chính. Ngôn ngữ được sử dụng là C++.Biên dịch bằng GNU g++ [8].

Bộ chương trình con được hiệu lực hóa bằng cách so sánh kết quả với chương trình mã nguồn mở đã được chứng nhận và sử dụng rộng rãi trên các phòng thí nghiệm trên thế giới, ROOT.Trong báo cáo này, 5 bộ số liệu đã được sử dụng để so sánh.

[1]Viện Nghiên cứu Hạt nhân Đà Lạt
[2] Trường Đại học Đồng Nai

2. Thuật toán và cách sử dụng chương trình

Thuật toán LEVENBERG-MARQUARDT

Xét bộ số liệu với n điểm thực nghiệm (X_i,Y_i), mô hình cần làm khớp là F(X,**α**), với **α** là vectơ tham số {$\alpha_1,\alpha_2,\alpha_3,\ldots,\alpha_m$). Theo đó:

$$Y_i=F(X_i,\alpha_1,\alpha_2,\alpha_3,\ldots,\alpha_m) \quad (1)$$

Để xác định các tham số tự do, ta sử dụng phương pháp bình phương tối thiểu [9]. Phương pháp này đòi hỏi phải xác định **α**sao cho là cực tiểu:

$$\chi^2 = \sum_i w_i.[y_i - F(x_i,\boldsymbol{\alpha})] \quad (2)$$

Trong đó là trọng số tương ứng với điểm số liệu thứ i. cực tiểu khi:

$$\frac{\partial \chi^2}{\partial \alpha_i} = 0 \quad (3)$$

Đối với các hàm tuyến tính, hệ m phương trình nói trên có thể được giải ra nghiệm xác định bằng phương pháp Gauss-Jordan.Tuy nhiên, với các bài toán phi tuyến, hệ phương trình trên không thể giải được. Khai triển F(X,**α**) theo chuỗi Taylor, ta thu được biểu thức dưới dạng ma trận:

$$\boldsymbol{b = M.d\alpha} \quad (4)$$

Trong đó M là ma trận [m m] mà:

$$M_{ij} = \sum_i w_i.\frac{\partial F(x_i,\alpha)}{\partial \alpha_j}.\frac{\partial F(x_i,\alpha)}{\partial \alpha_k} \quad (5)$$

Và

$$b_k = \sum_i w_i.\left(y_i - F(x_i,\alpha)\right).\frac{\partial F(x_i,\alpha)}{\partial \alpha_k} \quad (6)$$

là vectơ biến thiên của vectơ tham số .

Giải phương trình (3) cho phép xác định , từ đó xác định được mới. Thủ tục này lặp đi lặp lại nhiều lần cho tới khi hội tụ. Phương pháp LEVENBERG-MARQUARDT, bổ sung thêm vào thuật toán 2 tham số và , nhằm cải thiện khả năng hội tụ của quá trình khớp.

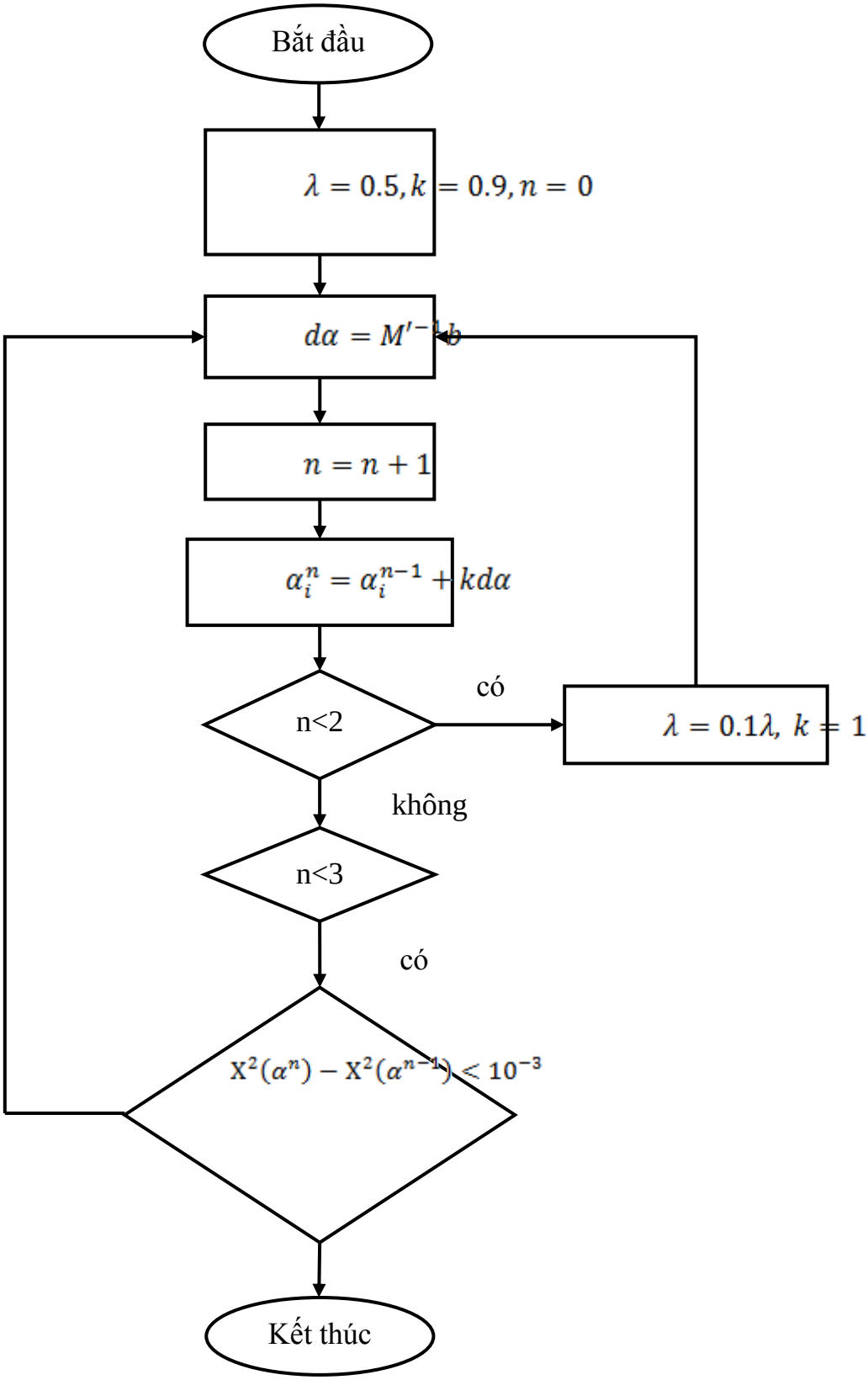
Thuật toán có thể được mô tả ngắn gọn, từng bước một như sau, lưu đồ thuật toán được đưa ra trong Hình 1:

1. Đặt , , n=0.
2. Xác định từ phương trình:

$$\boldsymbol{b = M'.d\alpha} \quad (7)$$

Với $M' =$, là ma trận đơn vị.

3. n=n+1
4. $\alpha_i^n = \alpha_i^{n-1}$.
5. Tính .
6. Nếu n<2, đi tới bước 9
7. Nếu n<3, đi tới bước 8
8. Nếu $\chi^2(\alpha^n) > \chi^2(\alpha^n$, trong đó ε : thì tiếp tục vòng lặp, nếu không, thoát ra khỏi vòng lặp.
9. Đặt $\lambda = 0.1\lambda$; Quay lại bước 2.



Hình 1. Lưu đồ thuật toán

Sử dụng chương trình con

Thủ tục làm khớp dữ liệu được thực hiện bởi hai chương trình con LSfit_NL (không có trọng số) và LSfit_NLW (có trọng số). Cú pháp khai báo như sau:

```
LSfit_NL(matrix X, matrix Y, int par_num, matrix par)
LSfit_NLW(matrix X, matrix Y, matrix W, int par_num, matrix par)
```

Trong đó X, Y là hai ma trận tương ứng với bộ số liệu thực nghiệm (X,Y), W là ma trận trọng số, par_num là số tham số tự do của mô hình làm khớp, par là ma trận tương ứng với giá trị ban đầu của tham số.

Mảng hai chiều hoặc một chiều có thể được chuyển thành ma trận (matrix) thông qua chương trình con array_to_matrix với cú pháp như sau:

```
array_to_matrix((double *)array, int row, int col);
```

array là mảng 1 chiều hoặc 2 chiều, row là số dòng, và col là số cột của ma trận tạo thành. Ví dụ, mảng hai chiều A[3][2] có thể được chuyển đổi thành ma trận MA[3][2] thông qua câu lệnh sau: MA = array_to_matrix((double*)A, 3, 2)

Mô hình làm khớp được khai báo bên trong chương trình con uf

```
double uf(double x, matrix par)
{
	double result;
	result = par.E[0]*exp(x/par.E[1]); //par.E[i] là tham số tự do thứ i, x là biến.
	return result;
}
```

Đoạn chương trình thực hiện tác vụ làm khớp bộ số liệu 15 điểm theo mô hình f(x)=a*exp(x/b) với a, b là các tham số tự do được đưa ra dưới đây:

```
double uf(double x, matrix par)
#include <iostream>
#include "matrix.h"
#include <math.h>

using namespace std;
{
	double result;
	result = par.E[0]*exp(x/par.E[1]); // par.E[0]=a; par.E[1]=b
	return result;
```

```
}
int main()
{
	double A[15]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
	double
B[15]={20,39,66,113,180,300,497,816,1346,2230,3674,6050,9
976,16454,27122};
	double C[15] = {0.222851, 0.160098, 0.122793,
0.094265, 0.074557 ,0.057718, 0.044859, 0.035006, 0.027256,
0.021178, 0.016497, 0.012857, 0.010012, 0.007796,
0.006072};
	double para[2]={10,2};
	matrix X = array_to_matrix(A,15,1);
	matrix Y = array_to_matrix(B,15,1);
	matrix W = array_to_matrix(C,15,1);
	matrix par = array_to_matrix(para,2,1);
	cout<<"No Weighted:"<<endl;
	Mprint(LSfit_NL(X,Y,2,par));
	par=array_to_matrix(para,2,1); //khởi tạo lại tham số ban
đầu
	cout<<"Weighted:"<<endl;
	Mprint(LSfit_NLW(X,Y,W,2,par));
return 0;
}
```

Mảng A, B, C lần lượt tương ứng với các ma trận X, Y, W.

Để hiệu lực hóa chương trình, kết quả tính toán thực hiện bởi chương trình trên nhiều bộ số liệu khác nhau với các mô hình liệt kê dưới đây được so sánh với chương trình ROOT.

Các mô hình làm khớp được thử nghiệm bao gồm:

- Mô hình hàm lũy thừa cơ số tự nhiên: f(x)=a*exp(x/b);

- Mô hình hàm gauss g(x)=A*exp(-(x-)^2/2), với A, , là các tham số tự do.

- Mô hình hàm gauss nằm trên một nền phông tương ứng với đa thức bậc 1: $f(x) = g(x) + a_1*x + a_0$

- Mô hình hai hàm gauss nằm chập lên nhau chồng trên một nền phông tương ứng với đa thức bậc 1: $f(x) = g_1(x) + g_2(x) + a_1*x + a_0$

- Mô hình ba hàm gauss chập trên nền phông tương ứng với đa thức bậc 1: $f(x)=f(x) = g_1(x) + g_2(x)+g_3(x)+ a_{1*}x + a_0$

3. Hiệu lực hóa chương trình thông qua so sánh với ROOT

Kết quả thu được bởi chương trình được so sánh với ROOT, phần mềm được sử dụng rộng rãi bởi nhiều phòng thí nghiệm trên thế giới. Kết quả so sánh với một số mô hình được trình bày trong Bảng 1.

Bảng 1. So sánh giá trị tham số làm khớp của chương trình với ROOT

Tham số	Gía trị ban đầu	Chương trình này	ROOT	Độ lệch giữa hai chương trình (%)
Hàm lũy thừa cơ số tự nhiên: y=a*exp(x/b)				
Không trọng số				
a	10	15,0015	15,0015	0
b	2	2,0000	2,0000	0
Có trọng số				
a	10	14,9828	15,0119	0,19
b	2	1,9997	2,0002	0,03
Hàm gauss: $y = A*exp(-(x-\mu)^2/\sigma^2)$				
Không trọng số				
A	80	99,7945	99,7951	0,00
μ	52	50,0195	50,0195	0,00
σ	20	10,0041	10,0041	0,00
Có trọng số				
A	80	99,3681	99,9418	0,58
μ	52	50,1327	50,0060	0,25
σ	20	10,0439	9,9759	0,68
Gauss + đa thức bậc 1: $y = A*exp(-(x-\mu)^2/\sigma^2)+ a1*x + a0$				
Không trọng số				
A	80	99,8306	99,8308	0,00
μ	52	50,0090	50,0090	0,00
σ	10	10,0198	10,0198	0,00
a1	2	2,0047	2,0047	0,00
a0	3	2,7612	2,7611	0,00
Có trọng số				
A	80	99,7179	99,9112	0,19
μ	52	50,0157	50,0015	0,03
σ	10	10,0227	10,0274	0,05
a1	2	2,0026	2,0076	0,25
a0	3	2,8558	2,5562	10,49

2 hàm gauss chập + đa thức bậc 1: $y = A*exp(-(x-\mu)^2/\sigma^2)+ a1*x + a0 + A1*exp(-(x-\mu1)^2/\sigma1^2)$				
Không trọng số				
A1	80	79,8317	79,8321	0,00
μ1	31	29,9800	29,9801	0,00
σ1	8	7,9845	7,9845	0,00
a1	2	0,9997	0,9997	0,00
a0	3	2,0129	2,0128	0,00
A2	90	99,9497	99,9500	
μ2	51	49,9717	49,9717	
σ2	10	12,0307	12,0307	
Có trọng số				
A1	80	79,6992	79,8282	0,16
μ1	31	29,9754	29,9798	0,01
σ1	8	7,9896	7,9745	0,19
a1	2	0,9999	0,9997	0,03
a0	3	2,0053	2,0195	0,71
A2	90	99,8090	99,9412	0,13
μ2	51	49,9667	49,9676	0,00
σ2	10	12,0467	12,0371	0,08
3 hàm gauss chập + đa thức bậc 1: $A1*exp(-(x-\mu1)^2/\sigma1^2)+ a1*x + a0 + A2*exp(-(x-\mu2)^2/\sigma2^2)+ A3*exp(-(x-\mu3)^2/\sigma3^2)$				
Không trọng số				
A1	75	80,1944	80,2094	0,02
μ1	25	31,0066	31,0096	0,01
σ1	10	7,9983	8,0012	0,04
a1	1	1,9997	1,9998	0,00
a0	2	2,9995	2,9983	0,04
A2	100	90,0007	89,9736	0,03
μ2	45	50,9908	50,9791	0,02
σ2	8	9,9805	9,9628	0,18
A3	210	200,2460	200,4460	0,10
μ3	65	60,9977	60,9968	0,00
σ3	8	4,9986	5,0010	0,05
Có trọng số				
A1	75	80,6545	80,3449	0,38
μ1	25	31,1255	31,0127	0,36
σ1	10	8,1015	8,0189	1,02
a1	1	2,0004	2,0020	0,08
a0	2	2,9873	2,8098	5,94
A2	100	88,8030	90,0044	1,35

μ2	45	50,5029	50,9623	0,91
σ2	8	9,2869	9,9319	6,95
A3	210	208,4220	200,7630	3,67
μ3	65	60,9426	60,9965	0,09
σ3	8	5,0936	5,0069	1,70

4. Kết quả

Mô hình đầu tiên được sử dụng để so sánh là mô hình hàm lũy thừa cơ số tự nhiên. Mô hình hàm lũy thừa là dạng mô hình điển hình nhất khi tiến hành thủ tục làm khớp phi tuyến, do tham số ảnh hưởng rất mạnh tới giá trị của hàm. Chỉ một lượng nhỏ thay đổi trong tham số cũng khiến giá trị của hàm thay đổi một lượng lớn.Kết quả trong Bảng 1 cho thấy, khi làm khớp không trọng số, chương trình hội tụ về giá trị tham số hoàn toàn giống với ROOT. Đối với quá trình làm khớp có trọng số, kết quả thulệch so với ROOT một lượng nhỏ hơn 0.2%.

Các mô hình gauss, gauss trên nền đa thức bậc một, chập 2 hàm gauss trên nền đa thức bậc 1, và chập 3 hàm gauss trên nền đa thức bậc một đều cho kết quả tương đồng với ROOT. Độ chênh lệch của giá trị tham số làm khớp thu được bởi chương trình với giá trị thu được từ ROOT phần lớn đều nhỏ hơn 1%. Chỉ có một số ít trường hợp, giá trị tham số làm khớp thu được bởi chương trình lệch so với ROOT cao hơn 1%. Tuy nhiên trong các trường hợp đó, các tham số có độ lệch cao là các tham số có mức độ ảnh hưởng tới giá trị của hàm số rất nhỏ. Ví dụ như trường hợp tham số a0 thu được khi làm khớp với mô hình gauss trên nền đa thức bậc 1, độ lệch của chương trình với ROOT là 10,4% (2,8585 so với 2,5562). Mặc dù độ lệch cao, nhưng ảnh hưởng của tham số này tới giá trị của hàm là rất nhỏ.

Kết quả có sự tương đồng cao giữa chương trình với ROOT khi áp dụng vào các mô hình chập gauss cho thấy, chương trình hoàn toàn đáp ứng tốt bài toán tách đỉnh chập, vốn rất phổ biến khi phân tích phổ gamma.

5. Kết luận

Chương trình làm khớp có kết quả có độ tương đồng cao với ROOT. Cấu trúc của chương trình đơn giản, thuần túy chỉ sử dụng các thư viện có sẵn của C++, thuận tiện cho việc nhúng vào các chương trình con khác.

Chương trình rất thích hợp để tích hợp vào các chương trình phân tích phổ tự thiết kế, qua đó giúp giảm chi phí mua phần mềm phân tích đắt tiền, với các tính năng ít hoặc không bao giờ được sử dụng. Ngoài ra, việc dễ dàng chỉnh sửa mã nguồn, giúp người dùng dễ dàng xây dựng các mô-đun chuyên biệt nhằm thực hiện các tác vụ theo yêu cầu cụ thể một cách thuận tiện và nhanh chóng.

TÀI LIỆU THAM KHẢO

1. [Online]. Available: http://www.originlab.com/.
2. [Online]. Available: http://scidavis.sourceforge.net/.
3. [Online]. Available: https://root.cern.ch/.
4. [Online]. Available: https://www.r-project.org/.
5. [Online]. Available: http://www.mathworks.com/products/matlab/.
6. [Online]. Available: http://www.gnuplot.info/.
7. Gill, P. R.; Murray, W.; and Wright, M. H. "The Levenberg-Marquardt Method." §4.7.3 in Practical Optimization. London: Academic Press, pp. 136-137, 1981..
8. [Online]. Available: http://www.mingw.org/.
9. Rao, C. R.; Toutenburg, H.; et al. (2008). Linear Models: Least Squares and Alternatives. Springer Series in Statistics (3rd ed.). Berlin: Springer. ISBN 978-3-540-74226-5

DEVELOPMENT OF SUBROUTINE FOR DATA FITTING WITH VARIOUS MODELS

ABSTRACT

Currently, there are many computer programs, which allow users to fit experimental data to any mathematical models. However, these programs either do not give users their source codes (commercial software) or have complicated libraries (open source software). Consequently, using their libraries to form homemade software becomes a difficult task, and even impossible, in case of commercial software. This work presents a group of sub-programs, written in C++, which permit users to fit experimental data to any mathematical models, including weighted fit and non-weighted fit. The sub-programs are packaged in one file with only 438 code lines; hence, make it easy to develop programs based on these sub-programs. The quality of these sub-programs was proved by comparing with ROOT.

Keywords:*Fitting C++ code, Levenberg–Marquardt algorithm*