Física Numérica

Tarea #5

D. A. Vázquez Gutiérrez

Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional,Unidad Profesional .ªdolfo López Mateos", Zacatenco, Edificio 9, Col. San Pedro Zacatenco, C.P. 07730 del. Gustavo A. Madero, Ciudad de México, México

email: dvazquezg1600@alumno.ipn.mx

2 de junio de 2024

1. Interpolación de Lagrange

La idea general de la propuesta de Lagrange es encontrar *polinomio de grado mínimo* que pase por todos los puntos tabulados, asi:

$$g(x) \approx \sum_{i=1}^{n} g_i L_i(x) \tag{1}$$

donde:

$$L_i(x) = \prod_{j(\neq i)=1}^{n} \frac{x - x_j}{x_i - x_j}$$

(a) Escribimos el siguiente programa para implementar el método de Interpolación de Lagrange :

```
import sympy  #biblioteca que reune toda las caracteristicas dwe un sistema de algebra computacional
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd  #Biblioteca con la que importamos los datos
from pylab import *

#--------Carga de Datos ----------

data = pd.read_csv('Datos PROBLEMA 1.txt', delim_whitespace=True) # delim_whitespace=True para manejar los espacios como delimitadores.
#Asi como tenemos los datos 'i=' es tratado por 'pandas' como q
E = data.iloc[0, 1:].astype(float).values
f = data.iloc[1, 1:].astype(float).values
sigma = data.iloc[2, 1:].astype(float).values

divNo=5

#data.iloc[1, 1:] selecciona la segunda fila (indice 1) excluyendo el primer valor (que es el encabezado de la fila). Luego, se convierte a tipo float y se extrae como un array de valores.
x=E
y=f
#--------Funciones---------------

def arg_prod(i, j):
    """ Argumento de la productoria de las bases polinomicas de Lagrange.
    """
    # Variable simbolica
    x_sim = sympy.symbols('x') #aqui se hace uso de las propiedades algebraicas de sympy
    return (x_sim-x[i]) / (x[j]-x[i]) if i != j else 1

def interpolacion_lagrage(x, y, num_puntos):
    """ Estima la curva generada por
```

```
    el polinomio de lagrange que
    interpola los puntos datos

    args:
        x (np.array): Datos del eje x
        y (np.array): Datos del eje y
        num_puntos (int): Numero de
    puntos estimados a partir del
    polinomio

    returns:
        Puntos (x, y) estimados a
    partir del polinomio encontrado.
    Tupla
    """
    # Variable simbolica
    x_sim = sympy.symbols('x')

    # Numero de puntos ingresados
    points = len(x) #longitud del
    vector X

    # Bases polinomicas Lj = [L1, L2,
    ..., Lk]
    Lj = []  #lista vacia que almacena
     las bases polinomicas
    for k in range(points):
        Lk = np.prod([arg_prod(i, k)
    for i in range(points)])  #usa la
    funcion previamente definida para
    calcular  un porducto de elementos
     con np.prod
        Lj.append(Lk)  #anade el
    elementdo LK a la base Lj

    # Polinomio de lagrange
    pol = sum(y*Lj)

   # Aqui, y * Lj realiza un producto
    elemento a elemento entre los
    valores y y las bases Lj.
    #sum(y * Lj) suma estos productos
    para construir el polinomio pol.

    # Se generan los datos x, y a
    partir del polinomio encontrado
    x_test = np.linspace(min(x), max(x
    ),num_puntos)
    y_pol = [pol.subs(x_sim, i) for i
    in x_test]
    #pol.subs(x_sim, i) sustituye el
    valor i en la variable simbolica
    x_sim dentro del polinomio pol y
    evalua el resultado.

    return x_test, y_pol


#-----------Aplicacion-----

#Obtenemos valores cada 5 Mev
NO=int(max(x)//divNo+1) #elint es para
     transformar en entero el numero ,
     el mas 1 es para incluir el 0 y
    el 200


# Puntos generados por el polinomio de
     Lagrange
x_test, y_pol = interpolacion_lagrage(
    x, y,NO)


# Grafica
# Configuracion para utilizar LaTeX
plt.rc('text', usetex=True)
plt.rc('font', family='serif')

fig, ax = plt.subplots(figsize=(8, 6),
     dpi=300)  # tamano de la figura y
     calidad
# Datos originales
#ax.scatter(x, y, color='red', marker
    ='o', )
# Plot de la interpolacion de Lagrange
ax.plot(x_test, y_pol, 'o', color='
    blue', label='Lagrange',markersize
    =3, linewidth=2)

plt.errorbar(E, f, yerr=sigma, fmt='or
    ', capsize=5,label='Datos')


# Configuracion de leyenda
ax.legend(loc='best')

# Etiquetas de los ejes
ax.set_xlabel(r'$E \, (\mathrm{MeV})$'
    , fontsize=14)
ax.set_ylabel(r'$f(E) \, (\mathrm{MeV
    })$', fontsize=14)

# Titulo de la grafica
ax.set_title(r'\textbf{Distribucion de
     $f(E)$ vs Energia de Resonancia}'
    , fontsize=16)

# Cuadricula
ax.grid(True, which='both', linestyle=
    '--', linewidth=0.5)

# Lineas de los ejes
ax.axhline(0, color='black', linewidth
    =0.5)
ax.axvline(0, color='black', linewidth
```

```
    =0.5)

# Ajustar diseno para que todo quepa
fig.tight_layout()

# Mostrar la grafica
plt.show()
#---------------------------

# Ejemplo de uso
N=2000
x_op, y_op = interpolacion_lagrage(x, y, N)
error=0.5
E_res = x_op[np.argmax(y_op)]


f_m = max(y_op) / 2

def encontrar_puntos_cercanos(x_op, y_op, E_res, f_m):
    puntos_por_detras = None
    puntos_por_delante = None
    distancia_minima_por_detras = float('inf')
    distancia_minima_por_delante = float('inf')

    # Recorrer la lista de energias interpoladas
    for i in range(len(x_op)):
        # Calcular la distancia entre la energia interpolada y E_res
        distancia1 = abs(x_op[i] - E_res)
        distancia2= abs(x_op[i] - E_res)

        # Verificar si la energia interpolada esta por detras de E res
        if i < np.argmax(y_op) and distancia1 < distancia_minima_por_detras:
                if abs(f_m-y_op[i])<error:
                    puntos_por_detras = x_op[i]

        distancia_minima_por_detras = distancia1
                #print(i)

        # Verificar si la energia interpolada esta por delante de E_res
        elif i > np.argmax(y_op) and distancia2 < distancia_minima_por_delante:
                if abs(f_m-y_op[i])<error:
                    puntos_por_delante = x_op[i]

        distancia_minima_por_delante = distancia2



    return abs(puntos_por_detras-puntos_por_delante)

# Ejemplo de uso
gamma= encontrar_puntos_cercanos(x_op, y_op, E_res, f_m)


gammat=55
Ert=78

print(' Energia de resonancia', E_res, '| Error absoluto ', abs(Ert-E_res), 'Error Relativo ' , abs((Ert-E_res)/Ert))
print(' Gamma = ', gamma ,'| Error absoluto ', abs(gammat-gamma), ' Error Relativo ' , abs((gammat-gamma)/gammat))
```

Código 1: Programa que ajusta un polinomio a un conjunto de n puntos utilizanso el algortimo de Lagrange . NOMBRE : "Lagrange.py"

Donde empleamos las ecuación (1) y principalmente la biblioteca *sympy* la cual nos proporciona características de un sistema algebraico , que es justo lo que requerimos al utilizar el algoritmo de Lagrange , debido a su naturaleza polinómica.

Por otra parte añadimos una parte del codigo exclusivamente para darle una naturalidad y conexion con el ambiente $LATEX$ a la grafica , y por ultimo , desarrollamos una funcion que nos ayude a encontrar los los valores de una funcion dado un valor , por la derecha y por la izquierda de este, el cual nos ayudo a encontrar la funcion gamma que mas adelante se vera.

(b) Por el momento , cargamos los valores de la figura 1 en el programa usando *pandas*, de tal

$i=$	1	2	3	4	5	6	7	8	9
$E_i\,(MeV)$	0	25	50	75	100	125	150	175	200
$f\,(E_i)\,(MeV)$	10.6	16.0	45.0	83.5	52.8	19.9	10.8	8.25	4.7
$\sigma_i\,(MeV)$	9.34	17.9	41.5	85.5	51.5	21.5	10.8	6.29	4.14

Figura 1: Datos de experimento de resonancia de Breit-Wigner

forma que obtenemos la grafica de la figura 2, donde hacemos que se hagan saltos cada $5MeV$, o sea , 4 interpolaciones .

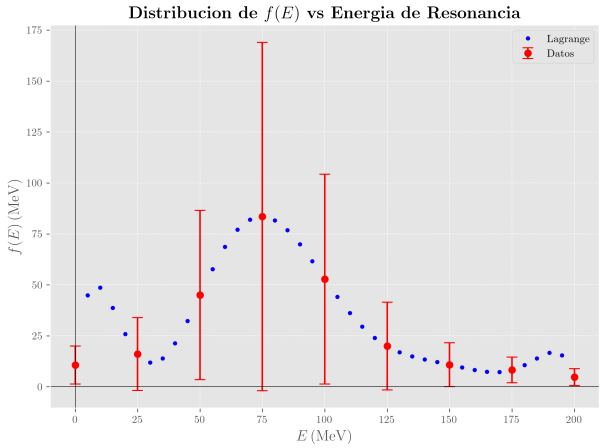

Figura 2: Grafica de experimento de resonancia de Breit-Wigner con ajuste polinómico utilizando algoritmo de Legendre

(c) Empleando la ultima parte del codigo donde encontramos el valor de E para el cual $f(E)$ se hace maximo :

$$E_{r_{Lgd}} = 74{,}537MeV \tag{2}$$

Así como el valor de Γ el cual es igual al grosor o diferencia de energía , entre los dos valores de esta que tienen como distribución $f(E)$ la mitad de la máxima, esto nos da

$$\Gamma_{Lgd} = 57{,}229MeV \tag{3}$$

Que con los valores teóricos , nos da la informacion contenida en el cuadro 1 .

Cuadro 1: Errores de E_r y Γ a partir de algortmo de Legendre (MeV)

Errores de E_r y Γ		
	Error absoluto	**Error relativo**
E_r	3.462731365682842	0.04439399186
Γ	2.22861430715	0.04052026013006

Donde podemos ver que el error relativo es muy pequeño , por lo que podemos decir que la aproximacion mediante el algoritmo de Legendre es valido al menos para la obtención de Γ y E_r .

2. Interpolación vía Splines cubicos

Ahora haremos un proceso similar al del inciso anterior , pero ahora emplearemos el método de *Splines cúbicos* , empleando la biblioteca *SciPy.interpoleate*, con *CubicSpline*.

Como básicamente es el mismo programa , solo colocamos las partes modificadas :

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import pandas as pd  #Biblioteca con la
       que importamos los datos
5  from pylab import *
6  from scipy.interpolate import CubicSpline
7
8  #--------Carga de Datos ----------
9
10 data = pd.read_csv('Datos PROBLEMA 1.txt',
       delim_whitespace=True) #
```

```
    delim_whitespace=True para manejar los
     espacios como delimitadores.
#Asi como tenemos los datos 'i=' es
    tratado por 'pandas' como q
E = data.iloc[0, 1:].astype(float).values
f = data.iloc[1, 1:].astype(float).values
sigma = data.iloc[2, 1:].astype(float).
    values

divNo=5

#data.iloc[1, 1:] selecciona la segunda
    fila (indice 1) excluyendo el primer
    valor (que es el encabezado de la fila
    ). Luego, se convierte a tipo float y
    se extrae como un array de valores.
x=E
y=f
#--------Funciones---------------

cs=CubicSpline(x, y)

#-----------Aplicacion-----

#Obtenemos valores cada 5 Mev
NO=int(max(x)//divNo+1) #elint es para
    transformar en entero el numero , el
    mas 1 es para incluir el 0 y el 200

x_test=np.linspace(min(x),max(x),NO)
y_test=cs(x_test)
```

Código 2: Programa que ajusta un polinomio a un conjunto de n puntos utilizando Splines Cubicos . NOMBRE : "SlinesCubicos.py"

Entonces, conseguimos la grafica de la Figura 3.

Podemos notar que en esta , la grafica no tiene tantas fluctuaciones como si las tiene el algoritmo de Legendre. Empleando la ultima parte del código donde encontramos el valor de E para el cual $f(E)$ se hace maximo :

$$E_{r_{SC}} = 76{,}238 MeV \tag{4}$$

Así como el valor de Γ el cual es igual al grosor o diferencia de energía , entre los dos valores de esta que tienen como distribución $f(E)$ la mitad de la máxima, esto nos da

$$\Gamma_{SC} = 57{,}929 MeV \tag{5}$$

Que con los valores teóricos , nos da la informacion contenida en el cuadro 2 .

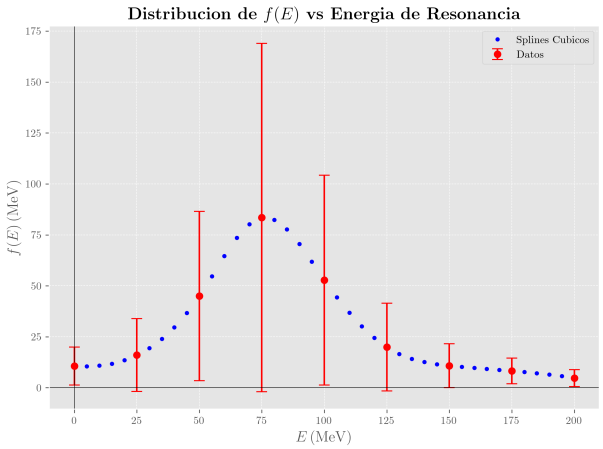

Figura 3: Grafica de experimento de resonancia de Breit-Wigner con ajuste polinómico utilizando Splines Cubicos

Cuadro 2: Errores de E_r y Γ apartir de Splines Cubicos (MeV)

Errores de E_r y Γ		
	Error absoluto	**Error relativo**
E_r	1.76188	0.02256
Γ	2.928964482	0.0532538

Donde aunque podemos ver que el error relativo es muy pequeño para Γ y E_r comparado con los valores teóricos, el error en la Energía de resonancia es ligeramente mejor ,mas sin embargo la Γ tiene ligero mayor error .

3. Ajuste a fórmula de resonancia de Breit-Wigner

(a) Para la teoría indica que la fórmula de Breit-Wigner debe ajustara los datos de los dos ejercicios anteriores:

$$f(E) = \frac{f_r}{(E - E_r)^2 + \Gamma^2/4} \tag{6}$$

Ahora queremos , utilizando esta formula teórica, encontrar los valores de los parametros E_r, f_r y Γ . Hacemos , por comodidad el siguiente cambio de variable:

$$g(x) = \frac{a_1}{(E - a_2)^2 + a_3}$$

Donde :

- $a_1 = f_r$
- $a_2 = E_R$
- $a_3 = \frac{\Gamma}{4}$

Para lograr esto , hacemos un ajuste por minimos cuadrados , donde nuestro objetivo es hacer que χ^2 sea minimo con respecto a los parametros , sea esto :

$$\frac{\partial \chi^2}{\partial a_m} = 0 \tag{7}$$

Para todo $m\epsilon[1, N]$ con N el numero máximo de parámetros y :

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - g(x_i; a_1, a_2, ..., a_N))^2}{\sigma_i^2}$$

Entonces buscamos :

$$\sum_{i=1}^{N} = \frac{(y_i - g(x_i))}{\sigma_i^2} \frac{\partial g(x_i)}{\partial a_m} = 0 \tag{8}$$

Entonces , encontramos las siguientes ecuaciones:

$$\frac{\partial g}{\partial a_1} = \frac{1}{(x - a_2)^2 + a_3)}$$

$$\frac{\partial g}{\partial a_2} = \frac{-2a_1(x - a_2)}{((x - a_2)^2 + a_3)}$$

$$\frac{\partial g}{\partial a_3} = \frac{-a_1}{((x - a_2)^2 + a_3)}$$

Entonces , siguiendo la ecuacion 7, equibalente a la ecuacion 8 , y empleando las ecuaciones encontradas anteriormente , encontramos las ecuaciones que minimizan la χ^2 :

$$f_1(a_1, a_2, a_3) = \sum_{i=1}^{9} = \frac{(y_i - g(x_i))}{\sigma_i^2((x - a_2)^2 + a_3)} = 0$$

$$f_2(a_1, a_2, a_3) = \sum_{i=1}^{9} = \frac{-2a_1(y_i - g(x_i))(x_i - a_2)}{\sigma_i^2((x - a_2)^2 + a_3)^2} = 0$$

$$f_3(a_1, a_2, a_3) = \sum_{i=1}^{9} = \frac{-a_1(y_i - g(x_i))}{\sigma_i^2((x - a_2)^2 + a_3)^2} = 0$$

(b) Notemos que estas ecuaciones no son lineales, por lo tanto , para resolver estas , hay que encontrar un vector (a_1, a_2, a_3) tal que haga que las ecuaciones anteriores se hagan cero.

Para ello , creamos un programa implementando el *Método de Newton Rapson Multidimencional*

```
import matplotlib.pyplot as plt
from pylab import *
import numpy as np
import numpy.linalg as npm
import math
import pandas as pd  #Biblioteca con
    la que importamos los datos

#--------Carga de Datos ----------

```

```
data = pd.read_csv('Datos PROBLEMA 1.txt', delim_whitespace=True) # delim_whitespace=True para manejar los espacios como delimitadores.
#Asi como tenemos los datos 'i=' es tratado por 'pandas' como q
E = data.iloc[0, 1:].astype(float).values
fo = data.iloc[1, 1:].astype(float).values
sigma = data.iloc[2, 1:].astype(float).values
sigma=sigma

# Definicion de  funciones f que minimizan \chi^2   --------------

#data.iloc[1, 1:] selecciona la segunda fila (indice 1) excluyendo el primer valor (que es el encabezado de la fila). Luego, se convierte a tipo float y se extrae como un array de valores.
x=E
y=fo


N=3  #dimencion  del problema a resolvedr
n=400 #Numero de espacios  de los parametros

a_min = [50000, 10, 10]  # Maximos y minimos de cada  dimencion  de a_i , CAMBIAR SI AUMENTAN LOS PARAMETOS N !!!
a_max = [100000, 100, 1000]  #ojo con los parametros de a_n , ya que hay que evitar indeterminaciones en las matrices

a = []

for i in range(N):
    a.append(np.linspace(a_min[i], a_max[i], n))

a = np.meshgrid(*a)   #generalizacion para cualquier n de a[0],a[1],a[2]= np.meshgrid(a[0],a[1],a[2])


#-----Eleccion de Semilla ----------

Ra = []

for i in range(N):
    Ra.append(a_max[i] - a_min[i])
Ra=np.array(Ra)

ta = []

for i in range(N):
    ta.append(Ra[i]/2)

ta=np.array(ta)   #semilla tentativa

Tx=[80400,70,560]

ta=Tx     #EL ERROR RADICABA EN LA SEMILLA ELEGIDA !!!!!  estaba obteniendo un buen resultado , para ENERGIAS NEGATIVAS!!!!!

#---Funciones  Particulares  ----------------



def g(a, x):
    a_1, a_2, a_3 = a
    g=a_1/((x - a_2) ** 2 + a_3)
    return g

def f_1(a, x, y, sigma):
    a_1, a_2, a_3 = a
    s = 0
    for i in range(len(x)):
        o = (y[i] - g(a, x[i])) / (((x[i] - a_2) ** 2 + a_3) * sigma[i] ** 2)
        s += o
    return s

def f_2(a, x, y, sigma):
    a_1, a_2, a_3 = a
    s = 0
    for i in range(len(x)):
        o = ((y[i] - g(a, x[i])) * ((x[i] - a_2))) / (((x[i] - a_2) ** 2 + a_3) ** 2 * sigma[i] ** 2)
        s += o
    return s

def f_3(a, x, y, sigma):
    a_1, a_2, a_3 = a
    s = 0
    for i in range(len(x)):
        o = ((y[i] - g(a, x[i]))) / (((x[i] - a_2) ** 2 + a_3) ** 2 * sigma[i] ** 2)
        s += o
    return s

```

```

f = [f_1, f_2, f_3]   #Vector de Funciones

#---Funciones Newton Rapson Multidimencional

da = [3.e-3,3.e-4,3.e-4] # MODIFICAR SI HAY MAS PARAMETROS

print(da)
Nmax = 100 # Parametros
Err0=5.e-12

def MapeoV(ta, x, y, sigma):
    F = []
    for func in f:
        F.append(func(ta, x, y, sigma))
    return np.array(F)  # Convertir a un array de numpy para operaciones vectoriales

# salto hacia adelante
def Mapeoplus(ta, i, j, da, x, y, sigma):
    tb = ta.copy() # Usar copy para no modificar ta original
    tb[j] += da[j]
    return f[i](tb, x, y, sigma)

# Funcion NEWTON RAPSON MULTIDMENCIONAL
def NewtonRM(ta, da, Err0, Nmax, x, y, sigma):
    N = len(ta)  # Numero de paramteros de semilla


    for it in range(Nmax + 1):
        print('Iteracion numero =', it)
        F = MapeoV(ta, x, y, sigma)
        print('Valor de F =', F)
        e0=F[0]
        e0=float(e0)
        e1=F[1]
        e1=float(e1)
        e2=F[2]
        e2=float(e2)

        # Verificar si la norma de F es menor que Err0
        if e0<Err0:
            if e1<Err0:
                if e2< Err0:
                    print("Condicion de error alcanzada. Terminando iteracion.")
                    break

#---------Preguntar POR que es que la busqueda menor que no se cumplia correctamente aqui?

        # Calculamos la jacobina  dfi/dxj
        df = np.zeros((N, N))
        for i in range(N):
            for j in range(N):
                df[i, j] = (Mapeoplus(ta, i, j, da, x, y, sigma) - f[i](ta, x, y, sigma)) / da[j]
                #print(Mapeoplus(ta, i, j, da, x, y, sigma),'|',i,j )
                #print(f[i](ta, x, y, sigma),'|',i,j)

        #print(df)   #para observar el comportamiento del Jacobino
        # Resolver el sistema lineal para encontrar la correccion
        try:
            df_inv = npm.inv(df)
        except np.linalg.LinAlgError:
            print("La matriz Jacobiana no es invertible. Terminando iteracion.")
            break


        # Resolver el sistema lineal para encontrar la correccion
        delta = np.dot(df_inv, -F)

        # Actualizar ta
        ta = ta + delta

    return ta



#El algoritmo de newton es preciso , pero nesesita  un valor inicial cercano a la raiz para ser realmente efectivo , por eso usaremos en conjunto biseccion  y newton rapson

ta=NewtonRM(ta, da, Err0, Nmax, x, y, sigma)
print('Entonces , tenemos que :\n')


```

```
print('fr=',ta[0],'\n')
print('Er=',ta[1],'\n')
print('Gamma=',math.sqrt(4*ta[2]),'\n')




x_test=np.linspace(min(x),max(x),n)
y_test=g(ta,x_test)



# Grafica
# Configuracion para utilizar LaTeX
plt.rc('text', usetex=True)
plt.rc('font', family='serif')

fig, ax = plt.subplots(figsize=(8, 6), dpi=300)  # tamano de la figura y calidad
# Datos originales
#ax.scatter(x, y, color='red', marker='o', )
# Plot de la interpolacion de Lagrange
ax.plot(x_test, y_test, '-', color='blue', label='Splines Cubicos', markersize=3, linewidth=2)
plt.errorbar(x, y, yerr=sigma, fmt='or', capsize=5,label='Datos')


# Configuracion de leyenda
ax.legend(loc='best')

# Etiquetas de los ejes
ax.set_xlabel(r'$E \, (\mathrm{MeV})$', fontsize=14)
ax.set_ylabel(r'$f(E) \, (\mathrm{MeV})$', fontsize=14)

# Titulo de la grafica
ax.set_title(r'\textbf{Distribucion de $f(E)$ vs Energia de Resonancia}', fontsize=16)

# Cuadricula
ax.grid(True, which='both', linestyle='--', linewidth=0.5)

# Lineas de los ejes
ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(0, color='black', linewidth=0.5)

# Ajustar diseno para que todo quepa
fig.tight_layout()

# Mostrar la grafica
plt.show()
```

Código 3: Programa que implementa medodo de Newton Rapson en varias dimenciones . NOMBRE : "NewtonRapMDIM.py"

El programa mostrado en el codigo 3 entonces es capaz de conducirnos a un ajuste a la ecuación 6 , minimizando las constantes para que ajusten de la manera mas apropiada a los datos de la figura 1.

Se hizo lo mejor para implementar correctamente el programa, sin embargo , aun hay una duda sobre el por que al definir F en la linea 120 , al emplear el codigo varias vece y diferentes formas, las sentencias if de las lineas 130 a 132 en veces no se ejecutaban y aunque , por ejemplo , se estableciera un error menor a $5e-9$ para todos las partes de F , al revisar el valor del vector F muchas de sus componentes tenian un valor mayor a el error establecido .

De igual forma , esta es la razón por la que el encontrar una solución lo suficientemente pequeña en función de una semilla aleatoria fue relativamente tedioso .

El autor piensa que el error se debe a razones numéricas , siendo números tan pequeños que la maquina los interpretaba como iguales . Pero si es así , ¿como podremos solucionar esta clase de problemas cuando requerimos tanta precision? . De cualquier forma , mediante la semilla $T_x = [80400, 70, 560]$, encontramos que :

$$f_r = 70873{,}0602(MeV)^3$$

$$E_r = 78{,}188MeV$$

$$\Gamma = 59{,}16MeV$$

que hacen :

$$f_1(a_1, a_2, a_3) = -3{,}048 \times 10^{-9}$$

$$f_2(a_1, a_2, a_3) = 2{,}0984 \times 10^{-13}$$

$$f_3(a_1, a_2, a_3) = -4{,}5062 \times 10^{-12}$$