

Глава 2. Интерполяция

Содержание главы:

- Введение
 - Задача интерполяции
 - Как осуществить линейную интерполяцию?
 - Как найти сплайн-аппроксимацию функции, заданной в дискретных точках?
 - Как интерполировать функцию, заданную дискретно?
 - Как производить сглаживание с помощью сплайн-функции?
 - Команда `fit_dat`
 - Команда `datafit`
 - Метод наименьших квадратов
-

Введение

Одна из наиболее часто возникающих задач состоит в обработке экспериментальных данных. Чтобы обработать (сгладить, аппроксимировать, интерполировать, экстраполировать и т.д.) экспериментальные данные, нужно иметь эти данные. Чаще всего экспериментальные данные представляют собой набор пар чисел (x_i, y_i) , где y_i – экспериментальные значения некоторой искомой функции, а x_i – значения аргумента. Пары экспериментальных данных перед дальнейшей обработкой желательно отсортировать по возрастанию значений x_i . Это можно сделать вручную или (при объемных массивах данных) автоматически, написав соответствующую программу для сортировки.

Пример программы такой сортировки.

Векторы x и y (в дальнейшем матрица v) соответствуют координатам экспериментальных данных. Хотим упорядочить экспериментальные точки по возрастанию значений координаты x .

```
x=[5 3 7 4 11];  
y=[50 30 70 40 110];  
// v=[x;y] - это матрица данных, являющаяся аргументов команд interp1n и др.  
[xx,k]=gsort(x,"g","i"); // gsort - команда сортировки  
yy=y(k);  
vv=[xx;yy] // упорядоченная матрица
```

При обработке данных эксперимента, разбросанных «как попало», часто хочется не рассматривать критерии оптимизации и просто провести линию (кривую или, на худой конец, ломаную, состоящую из прямых отрезков) непосредственно через точки. В среде Scilab есть для этого соответствующий инструментарий: средства линейной интерполяции (`interp1n`) и интерполяции сплайном (`interp` и `splin`).

Задача интерполяции

В интервале $[a, b]$ заданы $n+1$ опорных (узловых) точек $a \leq x_0 \leq x_1 \leq \dots \leq x_n \leq b$. Заданы $n+1$ значений y_i , соответствующих значению функции $y_i = f(x_i)$ в узловых точках. Необходимо найти многочлен $I_n(x)$ степени не больше n такой, что $I_n(x_i) = y_i$ для любого $0 \leq i \leq n$. Всегда имеется только один интерполяционный многочлен.

Интерполяцию применяют для того, чтобы вычислить значения функции между узловыми точками (интерполяция) или за интервалом узловых точек (экстраполяция).

Интерполяция на больших интервалах (т. е. с относительно небольшим количеством узловых точек) имеет дополнительные трудности:

- 1) точность при больших расстояниях между узловыми точками очень мала
- 2) интерполяционные многочлены высокого порядка на концах интервала значительно колеблются, что существенно искажают поведение функции, что особенно важно при последующем дифференцировании.

Как осуществить линейную интерполяцию?

С помощью команды **interp1n**

Синтаксис **[y]=interp1n(xyd,x)**

Параметры

xyd : двухстрочная матрица (ху координаты точек)

x : вектор (абсцисса)

y : вектор (значения по оси y)

Команда **interp1n** интерполирует функцию в интервалах между дискретными узлами, заданных с помощью матрицы **xyd** с помощью отрезков прямой и возвращает значение интерполирующей ломаной в дискретных точках, заданных вектором **x**.

Пример.

```
x=[1 10 20 30 40];  
y=[1 30 -10 20 40];  
xyd=[x;y]
```

Результат:

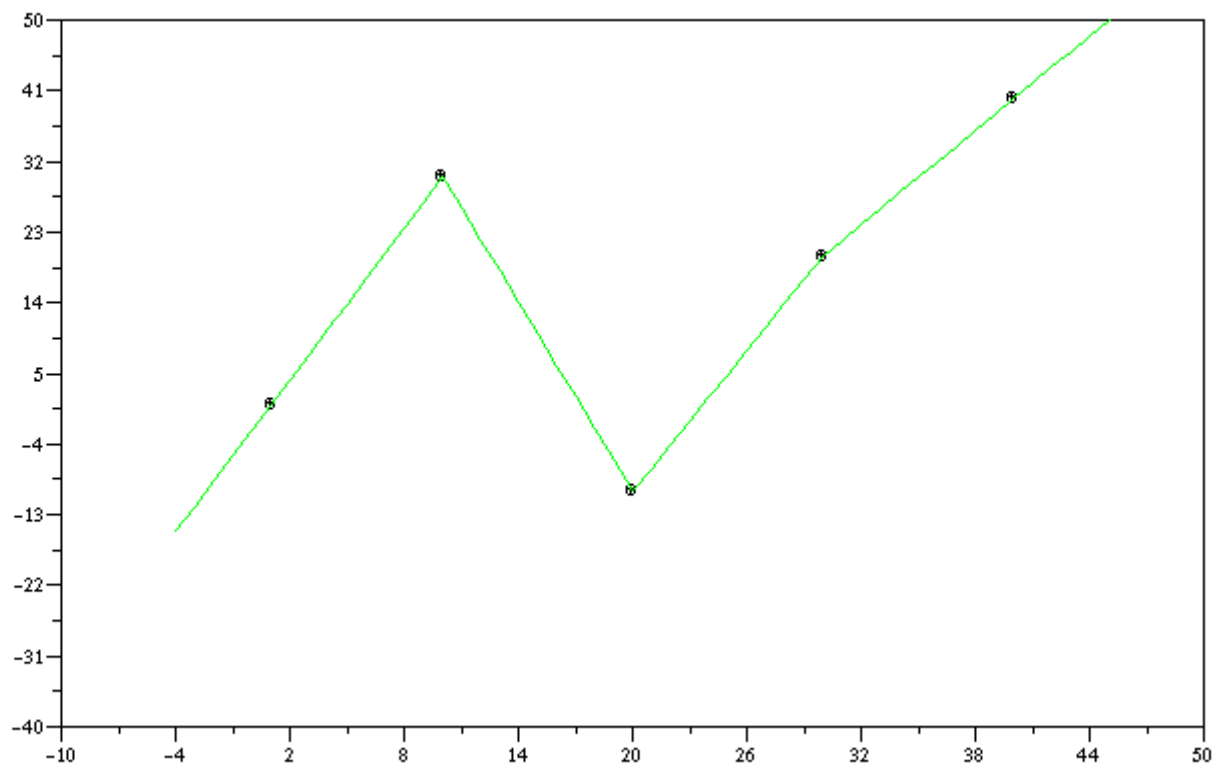
xyd =

```
! 1. 10. 20. 30. 40. !  
! 1. 30. - 10. 20. 40. !
```

```
plot2d(x',y',[-3],"011"," ",[-10,-40,50,50]);  
x_new=(-4:45);  
yi=interp1n(xyd,x_new);  
plot2d((-4:45)',yi',[3],"000");
```

Результат:

Будут определены значения интерполирующей функции в 50 точках от $x=-4$ до $x=50$ с шагом 1.



Можно найти интерполяционное значение функции в дискретной точке:

`z=interp1n(xyd,10.5);` Результат: `z= 28`.

Замечание: С помощью команды **interp1n** (хотя она и называется командой интерполяции) можно провести экстраполяцию данных и для значений x , лежащих вне интервала, заданного первой строкой матрицы `xyd`, но достоверность такой интерполяции очевидно невелика.

Как найти сплайн-аппроксимацию функции, заданной в дискретных точках?

С помощью команды **splin**.

Синтаксис

d=splin(x,f [, "periodic"])

Параметры

x : действительный вектор

f : действительный вектор того же размера, что и **x**

"periodic" : флаг в виде строки (ищем периодичный сплайн)

Компоненты вектора **x** должны быть заданы в возрастающем порядке. Аппроксимация производится с помощью сплайна не ниже кубического порядка. Подробно смотри **help splin**. Команда **splin** вектор производных сплайнов. Обычно эта команда используется в комбинации с командой **interp**.

Пример.

```
x=[0. 1.01 2.01 3. 4.02];
f=[0. 1. 3.9 8.75 16.5];
plot(x,f)
d=splin(x,f )
S=interp(0:0.1:4.2,x,f,d);
plot2d(x',f',-1); // изображаются только дискретные точки
plot2d((0:0.1:10)',S',2,'000')
```

Как интерполировать функцию, заданную дискретно?

С помощью команды **interp**. Это команда принимается обычно в комбинации с командой **splin** (сплайн). Стандартная задача аппроксимации экспериментальных значений с помощью кубической интерполяции.

Синтаксис

[f0 [f1 [f2 [f3]]]] = interp(xd,x,f,d)

Параметры

xd : действительный вектор

x, f, d : действительные вектора из сплайна (spline)

fi : вектора - результаты преобразования

Заданные три вектора (**x,f,d**) определяют сплайн-функцию (смотри команду **splin**) с $f_i = S(x_i)$, $d_i = S'(x_i)$. Эта функция вычисляет **S** (соответственно **S'**, **S''**, **S'''**) в точках $x_d(i)$.

x : вектор , состоящий из x_i . Необходимое условие: $x(1) < x(2) < \dots$

f : вектор из $S(x_i)$

d : вектор из $S'(x_i)$

f0 : вектор $[S(xd(1)), S(xd(2)), S(xd(3)), \dots]$

f(1 2 3) : вектор первой, второй и третьей производной **S** в точках $x_i = [xd(1), xd(2), \dots]$ и т.д.

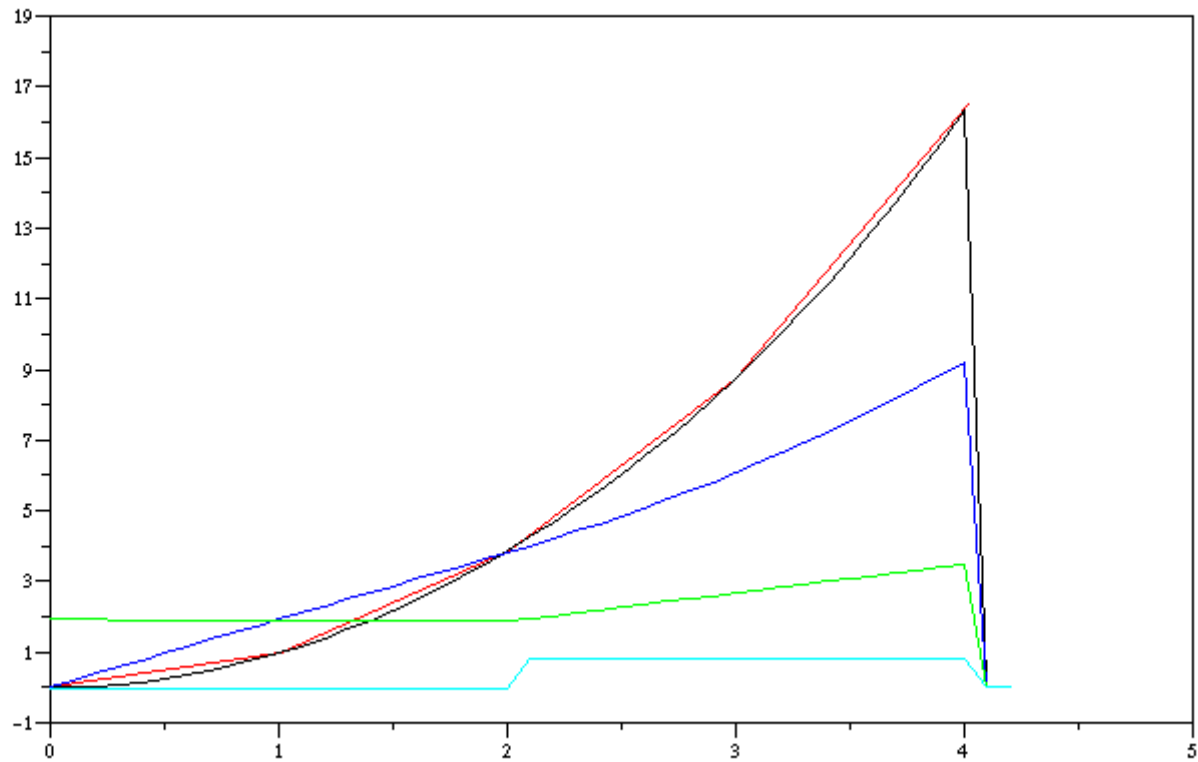
f1 = $[S'(xd(1)), S'(xd(2)), \dots]$

f2 = $[S''(xd(1)), S''(xd(2)), \dots]$

Пример.

```
x=[0. 1.01 2.01 3. 4.02];
f=[0. 1. 3.9 8.75 16.5];
plot(x,f);
d=splin(x,f );
xx=0:0.1:4.2;
s=interp(xx,x,f,d); //s будет совпадать с s0
[s0,s1,s2,s3]=interp(xx,x,f,d);
//s0 = интерполяционные значения функции
//s1 = первая производная
//s2 = вторая производная
//s3 = третья производная
plot2d(x,f,[5]);
plot2d(xx,[s0 s1 s2 s3]) // достаточно было бы plot2d(xx,s0)
```

Результат:



Красным цветом изображена ломаная, соединяющая экспериментальные точки. Видно, что интерполяционная кривая *s0* (черная линия) достаточно хорошо описывает экспериментальную кривую. Кривые *s1*, *s2* и *s3* описывают соответствующие производные моделируемой функции.

Замечание: В версии 2.7 (вариант Windows) при выполнении этого примера обнаружена ошибка: в то время, как *s1*, *s2* и *s3* являются векторами, *s0* является строкой. Поэтому для правильной работы приходится писать `plot2d(xx, [s0' s1 s2 s3])` вместо `plot2d(xx, [s0 s1 s2 s3])`. Вероятно, эта ошибка будет преодолена разработчиками.

Как производить сглаживание с помощью сплайн-функции?

С помощью команды **smooth**.

Синтаксис

[pt]=smooth(ptd [,step])

Параметры

ptd : (2 на n) действительный вектор

step : действительное число (шаг дискретизации по оси абсцисс)

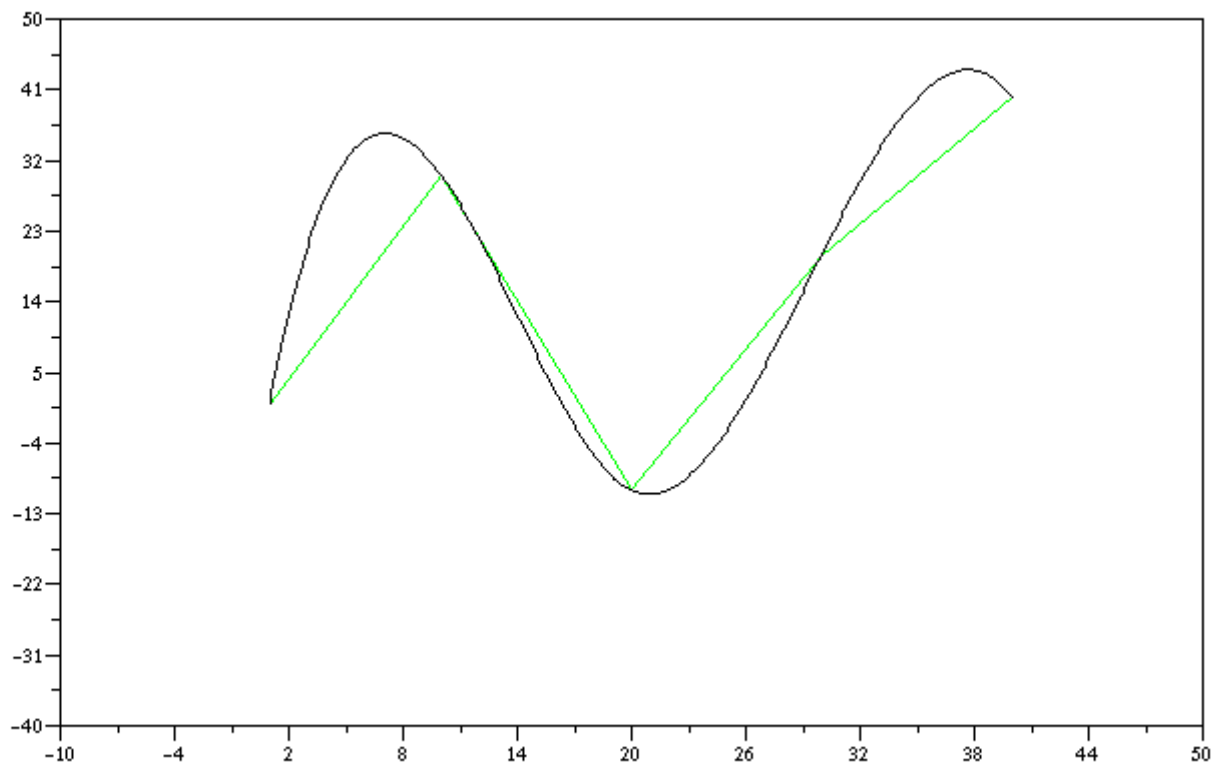
pt : (2 на n) действительный вектор

Команда вычисляет сглаживание с помощью сплайн-функции для функции, заданной таблицей значений в неравностоящих точках (узлах) плоскости. Координатами этих узлов являются **(ptd(1,i),ptd(2,i))**. Компоненты **ptd(1,:)** должны быть заданы в порядке возрастания. Необязательный параметр **step** принимает значение по умолчанию равное **abs(maxi(ptd(1,:))-mini(ptd(1,:)))/100**. Будет вычислены дискретные значения в промежуточных точках сглаженной функции с x-координатами в точках от **min(ptd(1,:))** до **max(ptd(1,:))** с шагом, равным **step**. Используются кубические сплайны аналогично команде **interp**.

Пример.

```
x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[3],"011"," ",[-10,-40,50,50]);
yi=smooth([x;y],0.1);
plot2d(yi(1,:) ',yi(2,:) ',[1],"000");
```

Результат:



Ломанной линией зеленым цветом изображен график реальных данных, соответствующий координатам $x=[1 \ 10 \ 20 \ 30 \ 40]$ и $y=[1 \ 30 \ -10 \ 20 \ 40]$. Гладкая черная линия соответствует результату сглаживания с помощью команды **smooth**.

Команда `fit_dat`

Команда `fit_dat` - параметрическое приближение экспериментальных данных, используется для "подгонки" данных к модельной функции. Является прототипом более совершенной команды `datafit`.

Синтаксис

`[p,err]=fit_dat(G,p0,Z [,W] [,pmin,pmax] [,DG])`

Параметры

G : Scilab функция ($e=G(p,z)$, где вектор e имеет размер ne на 1 , вектор p - np на 1 и вектор z -- nz на 1)

p0 : начальная гипотеза (размер np на 1)

Z : матрица $[z_1, z_2, \dots, z_n]$, где z_i (nz на 1) является i -тым измерением

W : весовая матрица размером ne на ne (необязательный параметр; значение по умолчанию равно 1)

pmin : нижняя граница p (необязательный параметр; размер np на 1)

pmax : верхняя граница p (необязательный параметр; размер np на 1)

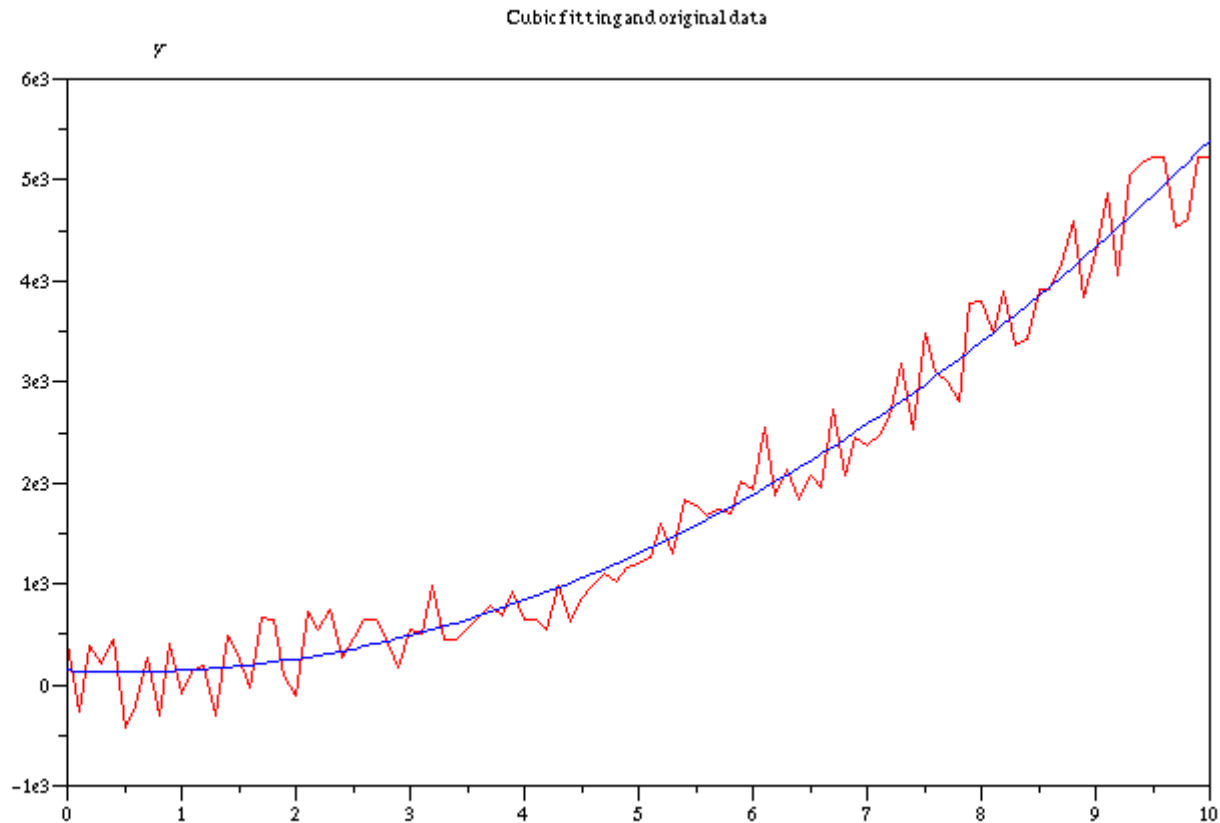
DG : partial of G wrt p (необязательный параметр; $S=DG(p,z)$, S : ne на np) Для заданной функции $G(p,z)$, команда находит наилучший вектор параметров p для аппроксимации $G(p, z_i)=0$ для набора векторов экспериментальных данных. Вектор p из условия минимизации $G(p, z_1)'WG(p, z_1)+G(p, z_2)'WG(p, z_2)+\dots+G(p, z_n)'WG(p, z_n)$.

Команда `fit_dat` выполняется в течении нескольких минут.

Пример.

```
x= [0:0.1:10];
y = 20+30*x+50*x^2;
dop = 1000*(rand(1,length(x))-0.5);
y1 = y + dop; // "экспериментальная" функция - парабола с небольшим "шумом"
z = [x;y1];
plot2d(x,y1,[5]);
// Подгонка экспериментальных данных z к кубическому полиному
deff(' [e]=G(a,z) ', 'e=z(2)-a(1)-a(2)*z(1)-a(3)*z(1)^2-a(4)*z(1)^3' );
a0 = [15;25;49;10];
[aa,err] = fit_dat(G,a0,z);
y_new=aa(1)+aa(2)*x+aa(3)*x^2+aa(4)*x^3;
plot2d(x,y_new,[2])
xtitle('Cubic fitting and original data','x','y')
```

Результат:



Красной линией изображены "экспериментальные данные", синей линией изображена модельная кривая.

Очень важное замечание: Если сравнить синтаксис вызова команд **fit_dat** и **datafit**, то видно, что порядок переменных в них различен. Сравните вышеизложенный пример с соответствующим примером на команду **datafit**: `[aa,er] = fit_dat(G,a0,z)` и `[aa,er] = datafit(G,z,a0)` Это очень плохо и, по-видимому, методичеки верно было бы во избежании путаницы никогда не употреблять команду **fit_dat**, как более несовершенную.

Команда datafit

Команда **datafit** осуществляет параметрическое приближение экспериментальных данных, используется для "подгонки" данных к модельной функции.

Команда **datafit** оптимально описывает экспериментальные данные полиномами не ниже второй степени.

Синтаксис

[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0],[mem]], [work],[stop],[in'])

Параметры

imp : скалярный аргумент, используемый для установки моды "слежения"(trace mode), т.е. при работе команды будут выдаваться отчеты о промежуточных результатах. При

значении **imp=0** не сообщается ничего (кроме сообщений об ошибках), **imp=1** начальный и финальный отчеты, Значение **imp=2** добавляет отчет для каждой итерации, при **imp>2** добавляет отчеты по линейному поиску. Ожидается, что эти отчеты пишутся на стандартный scilab-вывод.

G : функция-описатель следующего типа $\mathbf{e}=\mathbf{G}(\mathbf{p},\mathbf{z})$, где **e**: **ne** на 1, **p**: **np** на 1, **z**: **nz** на 1

DG : partial of G wrt p function descriptor (необязательный параметр); $\mathbf{S}=\mathbf{DG}(\mathbf{p},\mathbf{z})$, **S**: **ne** на **np**)

Z : матрица [**z_1**,**z_2**,...**z_n**], где **z_i** (**nz** на 1) является **i**-тым измерением

W : весовая матрица размером **ne** на **ne** (необязательный параметр; по умолчанию отсутствует)

contr : '**b**',**binf**,**bsup** с **binf** и **bsup** действительные вектора того же размера, что и **p0**. Параметры **binf** и **bsup** являются нижней и верхней границей **p**.

p0: начальное приближение (размер **np** на 1)

algo : '**qn**' или '**gc**' или '**nd**'. Эти строковые переменные используются для квази-Ньютоновского (по умолчанию), сопряженного градиента (conjugate gradient) или не дифференцируемого случая соответственно. Заметим, что '**nd**' не относится к границам по **x**.

df0 : действительный скаляр. Предполагается уменьшение **f** при первой итерации. (Значение по умолчанию **df0=1**).

mem : целое число, число переменных, используемых при аппроксимации Гессiana (Hessian), (**algo**='**gc**' или '**nd**'). Значение по умолчанию около 6.

stop : последовательность необязательных параметров, контролирующих сходимость алгоритма. Параметр **stop** принимает значения '**ar**',**nap**, [**iter** [,**epsg** [,**epsf** [,**epsx**]]]],

где
= '**ar**' : зарезервированное для правила остановки выполнения команды как одно из следующих:

- = **nap** : определяет разрешенное максимальное число обращений к **fun**.
- = **iter** : определяет разрешенное максимальное число итераций
- = **epsg** : пороговая величина нормы градиента
- = **epsf** : пороговая величина контролирующая уменьшение **f**
- = **epsx** : пороговая величина контролирующая изменение **x**. Этот вектор (возможно матрица) того же размера, что и **x0** может быть использован для масштабирования **x**.

"in" : зарезервированное ключевое слово для инициализации параметров, используемых в том случае, когда **fun** задана как Fortran-процедура.

p: вектор-столбец, является оптимальным решением

err : скаляр, наименьшая квадратичная ошибка.

Команда **datafit** используется для "подгонки" данных к модельной функции.

Для заданной функции $\mathbf{G}(\mathbf{p},\mathbf{z})$, эта функция находится как лучший вектор параметров **p** для аппроксимации $\mathbf{G}(\mathbf{p},\mathbf{z}_i)=0$ для заданной последовательности ветров

экспериментальных данных z_i .

Вектор p находится из условия минимизации

$$G(p, z_1)'WG(p, z_1) + G(p, z_2)'WG(p, z_2) + \dots + G(p, z_n)'WG(p, z_n)$$

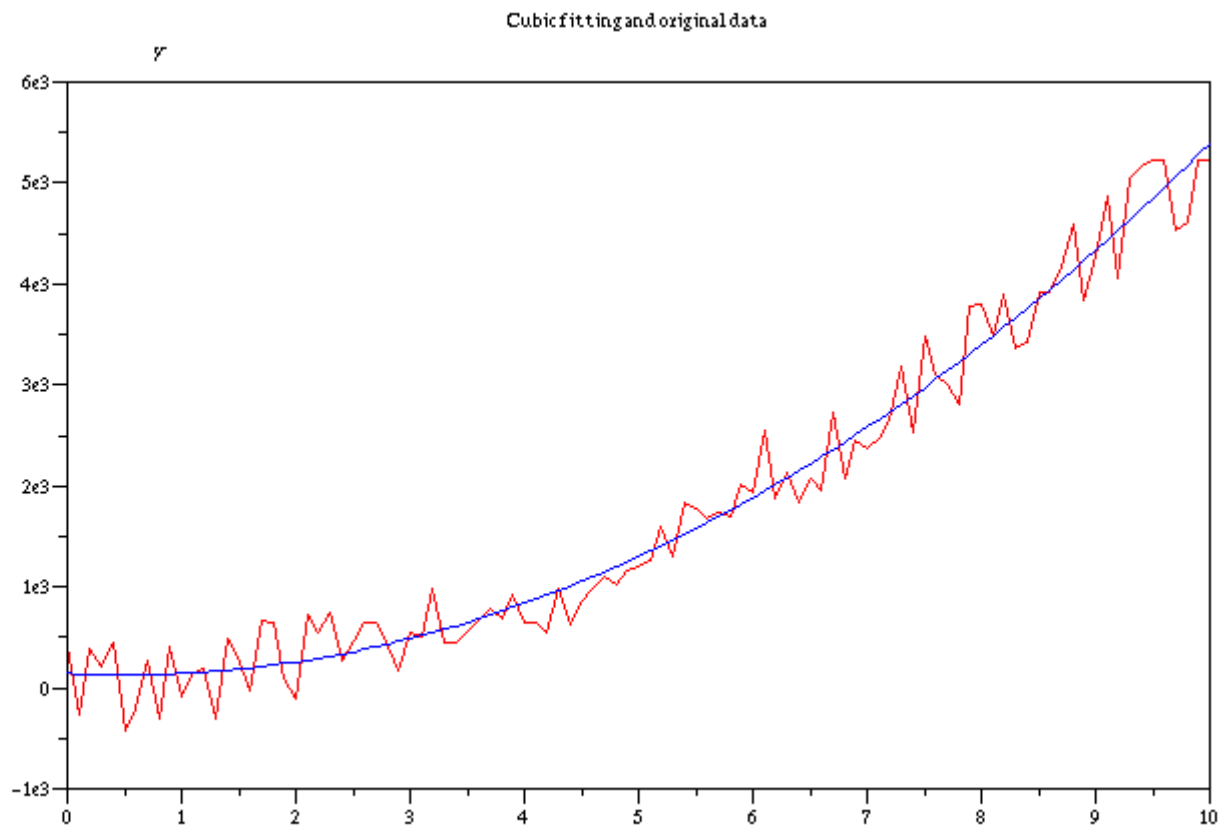
Команда **datafit** является улучшенной версией команды **fit_dat**, хотя может показаться слишком усложненной по синтаксису для начального освоения.

Замечание: Команда **datafit** выполняется в течении нескольких минут: имейте терпение.

Пример.

```
x = [0:0.1:10];  
y = 20+30*x+50*x^2;  
dop = 1000*(rand(1,length(x))-0.5);  
y1 = y + dop; //это "экспериментальная" функция - парабола на которую наложен  
"шум"  
plot2d(x,y1,[5]);  
  
z = [x;y1];  
// Будем искать модельную функцию в виде кубического полинома  
deff(' [e]=G(a,z) ', 'e=z(2)-a(1)-a(2)*z(1)-a(3)*z(1)^2-a(4)*z(1)^3');  
a0 = [15;25;49;10];  
[aa,er] = datafit(G,z,a0); // находим неизвестные параметры a(i)  
y_new=aa(1)+aa(2)*x+aa(3)*x^2+aa(4)*x^3;  
plot2d(x,y_new,[2])  
xtitle('Cubic fitting and original data','x','y')
```

Результат:



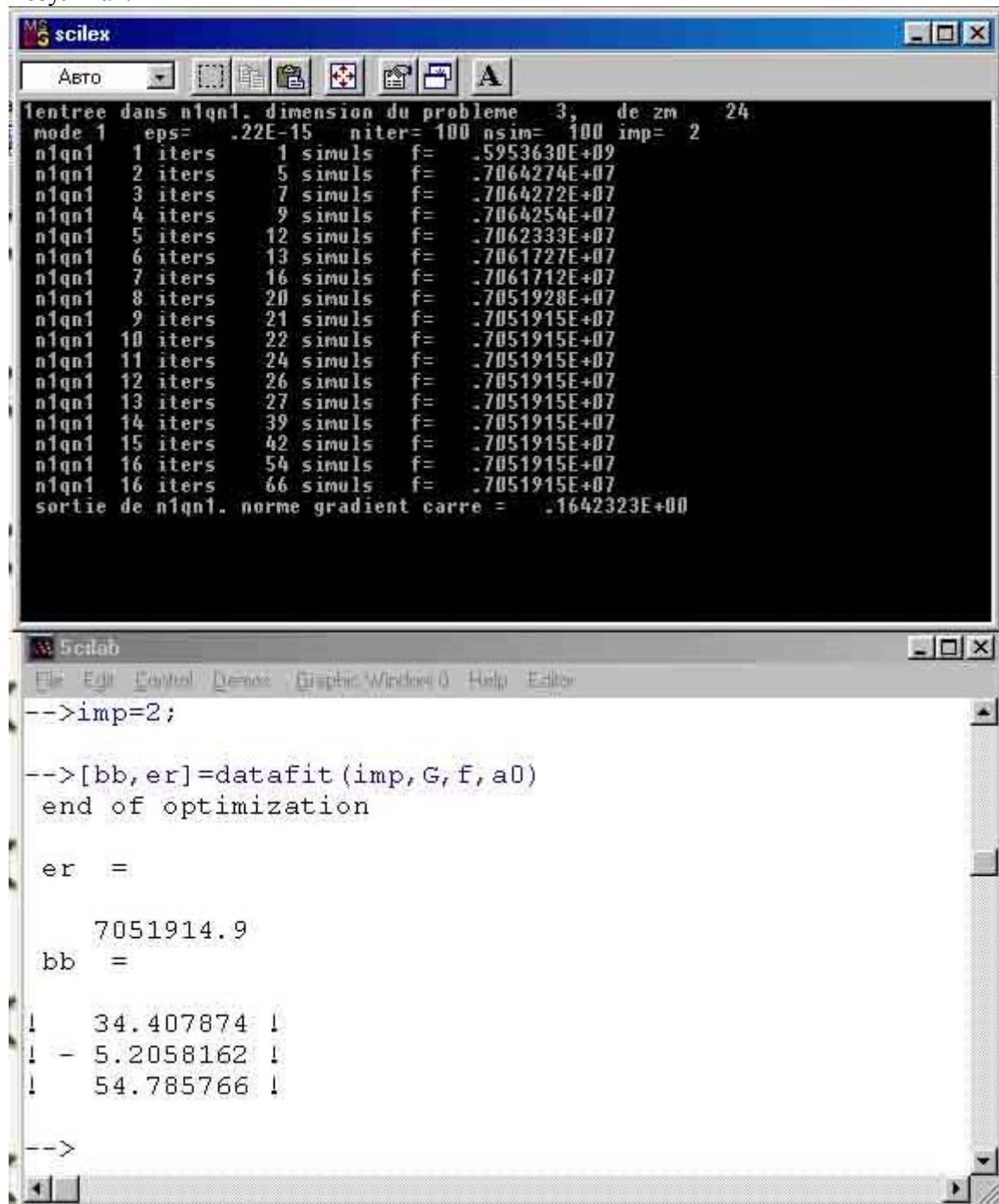
Красной линией изображены "экспериментальные данные", синей линией изображена модельная кривая.

При замене строки `[aa,er] = datafit(G,z,a0)` в приведенном выше примере на комбинацию из двух строк:

```
imp=2;  
[bb,er] = datafit(imp,G,f,a0)
```

мы получим дополнительные сведения о ходе проведения итераций. Это удобный режим для отладки. Заметьте, что эти сведения находятся не в главном окне Scilab, а в окне Scilex (оно черного цвета).

Результат:



The image shows two overlapping windows from the Scilab environment. The top window, titled 'scilex', has a black background and displays a detailed log of an optimization process. It includes parameters like 'mode 1', 'eps= .22E-15', 'niter= 100', 'nsim= 100', and 'imp= 2'. A table follows, listing iterations from 1 to 16, with columns for 'iters', 'simuls', and 'f=' values. The 'f=' values start at $-.5953630E+09$ and converge to $-.7051915E+07$. The final line shows 'norme gradient carre = .1642323E+00'. The bottom window, titled 'Scilab', has a white background and shows the command prompt with the executed commands: `-->imp=2;`, `-->[bb,er]=datafit(imp,G,f,a0)`, and `end of optimization`. Below the commands, the results are displayed: `er =` followed by `7051914.9`, and `bb =` followed by a column vector `[34.407874; -5.2058162; 54.785766]`.

```
lentree dans n1qn1. dimension du probleme 3, de zm 24  
mode 1 eps= .22E-15 niter= 100 nsim= 100 imp= 2  
n1qn1 1 iters 1 simuls f= -.5953630E+09  
n1qn1 2 iters 5 simuls f= -.7064274E+07  
n1qn1 3 iters 7 simuls f= -.7064272E+07  
n1qn1 4 iters 9 simuls f= -.7064254E+07  
n1qn1 5 iters 12 simuls f= -.7062333E+07  
n1qn1 6 iters 13 simuls f= -.7061727E+07  
n1qn1 7 iters 16 simuls f= -.7061712E+07  
n1qn1 8 iters 20 simuls f= -.7051928E+07  
n1qn1 9 iters 21 simuls f= -.7051915E+07  
n1qn1 10 iters 22 simuls f= -.7051915E+07  
n1qn1 11 iters 24 simuls f= -.7051915E+07  
n1qn1 12 iters 26 simuls f= -.7051915E+07  
n1qn1 13 iters 27 simuls f= -.7051915E+07  
n1qn1 14 iters 39 simuls f= -.7051915E+07  
n1qn1 15 iters 42 simuls f= -.7051915E+07  
n1qn1 16 iters 54 simuls f= -.7051915E+07  
n1qn1 16 iters 66 simuls f= -.7051915E+07  
sortie de n1qn1. norme gradient carre = .1642323E+00  
  
Scilab  
File Edit Control Demos Graphic Windows Help Editor  
-->imp=2;  
  
-->[bb,er]=datafit(imp,G,f,a0)  
end of optimization  
  
er =  
  
7051914.9  
bb =  
  
! 34.407874 !  
! - 5.2058162 !  
! 54.785766 !  
  
-->
```

Замечание: Интересно, что вычисления в пакетах Scilab разных версий (2.6 и 2.7) могут дать различные результаты. Версии 2.7 нужно гораздо меньшее число итераций для сходимости. Радует, что производитель работает над собой!

Метод наименьших квадратов

Метод наименьших квадратов является частным случаем применения команды **datafit**.

Пример 1.

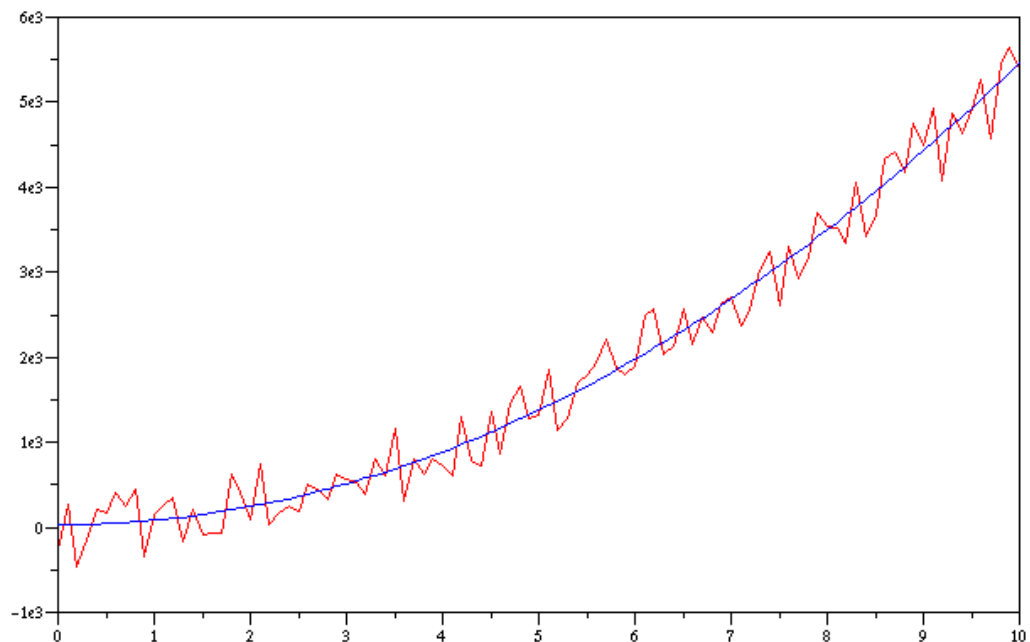
В качестве экспериментальной кривой зададим функцию, близкую к квадратичной: возьмем квадратичную кривую и наложим на нее с помощью генератора случайных чисел небольшой "шум".

```
x=[0:0.1:10];  
y=20+30*x+50*x^2;  
e=1000*(rand(1,length(x))-0.5); //это шум  
y1=y+e;  
plot2d(x,y1,[5]) ; // красный цвет - "экспериментальная кривая"
```

Будем считать узлы $(x, y1)$ модельными "экспериментальными" данными.

```
f=[x;y1]; //строим матрицу  
// определим функцию ошибки  
deff("[e]=G(a,z)", "e=z(2)-a(1)-a(2)*z(1)-a(3)*z(1)^2");  
a0=[1;1;1]; //ввод начальных параметров  
// найдем для модельной функции G значения коэффициентов разложения  
aa(1), aa(2) и aa(3)  
[aa,er]=datafit(G,f,a0);  
yy=aa(1)+aa(2)*x+aa(3)*x^2;  
plot2d(x,yy,[2])
```

Результат:



Красной линией изображены "экспериментальные данные", синей линией изображена модельная кривая, построенная с помощью метода наименьших квадратов.
Замечание: Команда **datafit** выполняется в течении нескольких минут: имейте терпение.

Пример 2. Разберемся на этом примере, что представляет собой значение параметра **er**, возвращаемое командой **datafit**.

```
x=[1 2 3 5 9];
y=[3 4 7 9 17];
plot2d(x, y, [5]);
f=[x;y];
deff("e]=G(a,z)", "e=z(2)-a(1)-a(2)*z(1)-a(3)*z(1)^2");
a0=[1;1;1];
[aa,er]=datafit(G,f,a0);
yy=aa(1)+aa(2)*x+aa(3)*x^2;
plot2d(x,yy,[2]);
e=(y-yy)^2; // это вектор ошибки приближения
d=sum(e) // Сумма элементов вектора. Она совпадает с er.
```

Практически получили аппроксимацию прямой линией, так как значение одного из элементов вектора коэффициентов **a** очень мало. Значение погрешности аппроксимации - эта величина **delta**, вычисляющаяся как корень квадратный из суммы квадратов отклонений, деленный на число измерений (данных).

```
n=sum(x);
delta=sqrt(n(2));
```

Мы убедились, что значение ошибки вычислений **er**, которую нам возвращает команды **datafit** равно сумме квадратов отклонения экспериментальных значений от значений модельной функции (иначе сумма квадратов остатков = sum of squared residuals =RSS). Для большого количества точек эта величина может быть велика.