Chapter 2

Least Square Fitting

Let us assume that $y = f_{c_1,\ldots,c_k}(x_1, \ldots, x_m)$ is a real-valued function of $(x_1, \ldots, x_m)$ which depends upon k parameters $c_1, \ldots, c_k$. These parameters are unknown to us. However, suppose that we can perform repeated experiments that for given values of $(x_1, \ldots, x_m)$ allows us to measure output values for y. How can we estimate the parameter $c_1, \ldots, c_k$ that best correspond with this information?

Let us assume that the experiment measuring the value $y = f_c(x)$ for specific input values $x = (x_1, \ldots, x_m)$ is repeated n-times. We will then obtain a system of equations

$$\begin{array}{rcl} f_{c_1,\ldots,c_k}(x_{11}, x_{12}, \cdots, x_{1m}) & = & y_1 \\ f_{c_1,\ldots,c_k}(x_{21}, x_{22}, \cdots, x_{2m}) & = & y_2 \\ \vdots & & \\ f_{c_1,\ldots,c_k}(x_{n1}, x_{n2}, \cdots, x_{nm}) & = & y_n \end{array}$$

where $x_{\imath\jmath}$ and $y_\imath$ are the measurements of $x_\jmath$ and y in the $\imath$ experiment. These experimental results gives us information about the unknown coefficients $c_\imath$'s.

Since we may perform the experiments as many times as we wish, we may end up with more relations of the type above than unknowns (that is, n is larger than k). The larger the n, the more information we have collected about the coefficients. However, even if the experiments are carried out with great care, they unavoidably will contain some error. The question remains: how may we estimate judiciously the coefficients $c_1, \ldots, c_k$ using the collected information about $y = f_c(x)$? What is the best fit?

A very common method to respond to this question is known as the *method of least-squares.* The idea is simple: per realization of the experiment, we measure the fitting error by the distance from the real number $f_c(x_1, x_2, \cdots, x_m)$ and the observed value of y. The best fit for the distance though will also lead to a best fit for the square of the distance. To avoid absolute values, we change our viewpoint slightly and measure the fitting error by $(f_{c_1,\ldots,c_k}(x_1, x_2, \cdots, x_m) - y)^2$.

The error function that considers all the information obtained from the n experiments is then

$$E(c_1, \ldots, c_k) = \sum_{i=1}^{n} ((f_{c_1,\ldots,c_k}(x_{i1}, x_{i2}, \cdots, x_{im}) - y_i)^2 ,$$

where $(x_{i1}, x_{i2}, \cdots, x_{im})$ and y_i are the i-th input for x and value of the measurement for y.

This turns out to be a function of $c = (c_1, \ldots, c_k)$. Mathematically, our best fit problem is now reduced to finding the value of c which produces a minimum for this error function E. The details of how this can be done depends intrinsically upon the assumed form of the function f, and its relation to the parameters $c_1, \ldots, c_k$.

2.1 Fitting a line to data

Suppose that we are in the situation described above, with $m = 1$, and that the function f is a polynomial of degree 1. Hence, the number of parameters is two, and for convenience we write

$$f(x) = mx + b ,$$

in order to interpret the parameters m and b as the slope and y-intersect of the graph of f. We have data points (x_1, y_1), ..., (x_n, y_n). The problem is now what values of m and b will best fit this data. Or put it simply, what is the straight line that best fit this information.

The contribution to the error function coming from the i-th piece of data, (x_i, y_i), is given by $(mx_i + b - y_i)^2$, and therefore, the total error is

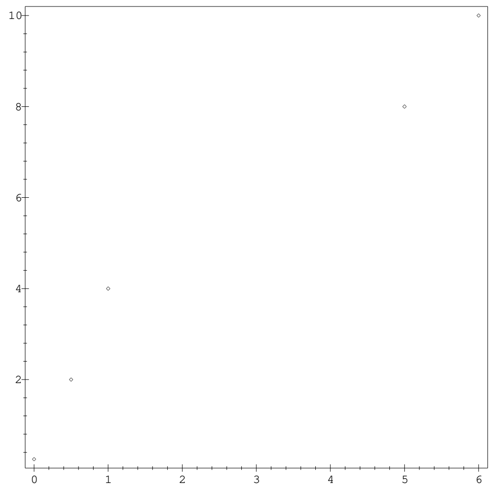
$$E(m, b) = \sum_{i=1}^{m} (mx_i + b - y_i)^2 .$$

The problem then becomes simply searching for the values of m and b where the error $E(m, b)$ achieves a minimum. Let us solve this problem using Maple.

First, let us type in some data to try to fit our line to:

```
>   pts := [ [0,0.25], [1/2,2], [1,4], [5,8], [6,10] ]:

>   plot(pts, style=point, axes=boxed);
```



In order to solve the problem at hand, we could use Maple's built-in regression package to find the answer, but that would not help much in explaining what is going on. However, let us do this anyway.

Since the least-squares package expects the x and y values in separate lists, we have to make a slight adjustment to our data:

```
> with(stats):
  fit[leastsquare[[x,y]]]([[ pts[i][1] $i=1..5], [ pts[i][2] $i=1..5]]);
```

$$y = 1.271370968 + 1.431451613x$$

This says that the best values of m and b are $m = 1.431451613$ and $b = 1.271370968$, respectively. Let us do this *by hand.*

First, we define the error function according to the data points to fit that we have. Notice again that this function is the square of the distance from the line to the data:

```
> E := (m,b)-> sum( ((m*pts[i][1]+b)-pts[i][2] )^2, i=1..5);
```

$$E := (m, b) \to \sum_{i=1}^{5} (m\, pts_{i1} + b - pts_{i2})^2$$

For example, had we guessed that the line $y = 2x + 1$ is the answer to our problem, we could compute the distance:

```
> E(2,1);
```

$$19.5625$$

However, decreasing m and increasing b produces a smaller value of E, so that guess cannot be the best fit:

```
> E(1.5,1.2);
```

$$3.125000000$$

We want to minimize the error function. Since minima of differentiable functions occur at critical points, we begin by solving for the values of (m, b) which annihilate the two partial derivatives. Notice that Maple happily computes partial derivatives:

```
> diff( E(m,b), m);
```

$$\frac{249}{2}m + 25b - 210$$

We may then ask Maple to solve the system of equations obtained by equating the two partial derivatives by

```
> solve( { diff( E(m,b), m)=0, diff( E(m,b), b)=0 }, {m,b});
```

m = 1.431451613, b = 1.271370968

Maple thus finds only one solution, and not surprisingly, it coincides with the solution found using the statistics package.

If we now want to use this values of m and b without retyping it, one way to do that is to use the command `assign(%)` to let b and m be given these constant values.[1]

```
> assign(%);
```

Maple executes this assignment silently, without reporting the assignment. However, now both m and b have the assigned values:

```
> m;
```

$$1.431451613$$

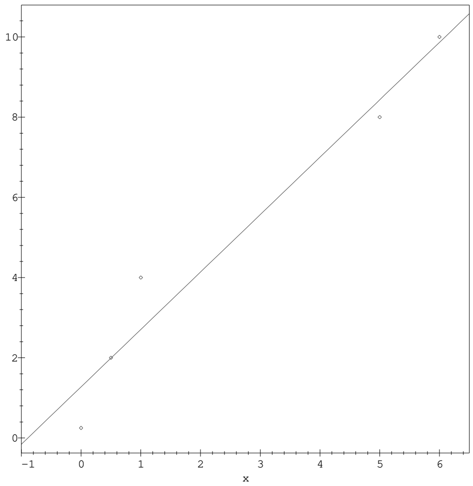
```
> E(m,b);
```

$$2.929334677$$

The value of $E(m, b)$ calculated above, is the value of E when (m, b) is the critical point found earlier.

We may visualize how good our fit is, using `plot` to display the data points and the fit line on the same graph. We load the plots package, so that we can use display:

```
> with(plots):
  lplot := plot(m*x+b, x=-1..6.5, axes=boxed):
  pplot := plot(pts, style=point):
  display({lplot, pplot});
```



[1]We are using maple's *ditto operator % to refer to the result of the previous command. In versions of maple prior to release 5 of Maple V, the double quote (") was used for the ditto operator.*

Notice that at this point, m and b have constant values previously assigned to them, the values producing the minimum of E up to 9 decimal places. Let us try to verify that our solution is indeed correct, via an argument which is independent of the one described above.

Geometrically, the error function E is the square of the Euclidean distance between $m\vec{x}+\vec{b}$ and the vector $\vec{y}$, where $\vec{x}=(x_1,x_2,\ldots,x_n)$, $\vec{b}=(b,b,\ldots,b)$ and $\vec{y}=(y_1,\ldots,y_n)$. Clearly then, the best fit is achieved when the coefficients m and b are chosen so that $m\vec{x}+\vec{b}$ is the orthogonal projection of the vector $\vec{y}$ onto the subspace of $\mathbb{R}^n$ spanned by $\vec{x}$ and $(1,1,\ldots,1)$. That is to say, the best fit occurs when the vectors $\vec{y}-(m\vec{x}+\vec{b})$ and $m\vec{x}+\vec{b}$ are perpendicular to each other. Let us verify this:

```
> sum( (pts[i][2]-m*pts[i][1]-b)*(m*pts[i][1]+b),i=1..5);
```

$$-.16\cdot 10^{-7}$$

The inner product is not quite zero, but the first non-zero digit of its decimal expansion occurs in the eighth decimal place. This is because the calculations were only done with finite precision.

As a final remark, we observe that if you now execute the command `diff( E(m,b),m )`, you will obtain an error. This is due to the fact that m and b are constants at this point, and not independent variables. If you would like to continue making use of the function $E(m,b)$ for general m and b, you will have to unassign the values of m and b first:

```
> m := 'm':
  b := 'b':
```

2.2 Fitting a cubic to data

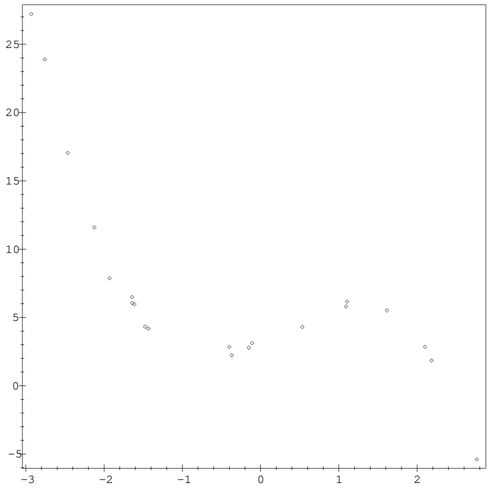
Let us now try to fit a cubic polynomial to some data. We do so directly, without using Maple's built in statistical package.

We randomly generate a set of 21 data points to be fitted to a cubic using a Maple program. (Don't worry about how this program works just now. We'll get into those details later).

```
> Seed:=readlib(randomize)():
  with(stats):
  myrand := (a,b) -> evalf(a+(b-a)*(rand()/10^12)):
  points_wanted := 21:
> cubic_pts := proc()
    local fuzz, a, b, c, d,s,x,i;
    a:= myrand(-3,3);
    b:= myrand(-3,3);
    c:= myrand(-3,3);
    d:= myrand(-10,10);
    if (myrand(-10,10) > 0) then s:=1 else s:=-1 fi;
    fuzz := [ random[normald[0,0.5]](points_wanted)];
    x    := [ random[uniform[-3,3]](points_wanted)];
    RETURN([seq([x[i],s*(a-x[i])*(b-x[i])*(c-x[i])+d+fuzz[i]],
              i=1..points_wanted)]);
  end:
```

Now we can put the data to be fitted into a list, and visualize the result with `plot`:

```
> data:=cubic_pts():
  plot(data, style=point, axes=boxed);
```



Never mind the way we generated the data points. As far as the problem to solve is concerned, we have a set of 21 data points which are to be fitted to a cubic polynomial. A polynomial function of degree n is determined by $n+1$ coefficients. So we define

```
> cub := (x,a,b,c,d)->a*x^3+b*x^2+c*x+d;
```

$$cub := (x,\ a,\ b,\ c,\ d) \to a\,x^3 + b\,x^2 + c\,x + d$$

and then define the error function:

```
> E:=(a,b,c,d)->sum((cub(data[i][1],a,b,c,d)-data[i][2])^2,i=1..nops(data));
```

$$E := (a,\ b,\ c,\ d) \to \sum_{i=1}^{\mathrm{nops}(data)} \left(cub(data_{i1}, a, b, c, d) - data_{i2}\right)^2$$

We have four parameters to determine, the coefficients of `cub`. We find its values by

```
> assign(solve({diff(E(a,b,c,d),a)=0,
                diff(E(a,b,c,d),b)=0,
                diff(E(a,b,c,d),c)=0,
                diff(E(a,b,c,d),d)=0},
               {a,b,c,d}));
```

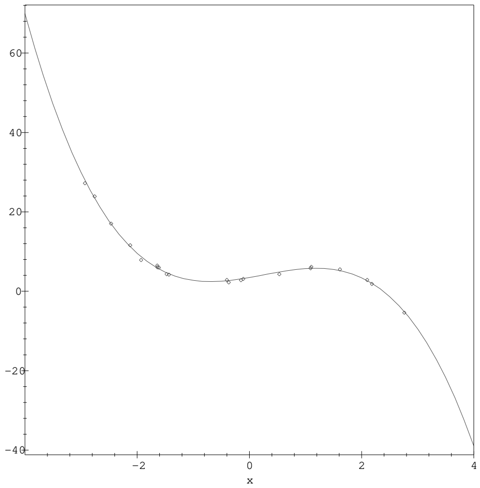
We do not see the answer, but the resulting values have already being assigned to the coefficients a, b, c, d.

```
> cub(x,a,b,c,d);
```

$$-1.006141915x^3 + .7505869176x^2 + 2.487673553x + 3.430571969$$

Finally, let's make a picture of the result, showing both the data and the fitted curve:

```
> with(plots):
  cplot := plot(cub(x,a,b,c,d), x=-4..4, axes=boxed):
  pplot := plot(data, style=point):
  display({cplot,pplot});
```



For comparison, let us find the answer using Maple's built-in statistical package:

```
>   with(stats):
    fit[leastsquare[[x,y],y=A*x^3+B*x^2+C*x+D]]
            ([[data[i][1]$i=1..nops(data)],[data[i][2]$i=1..nops(data)]]);
```

$$y = -1.006141912x^3 + .7505869193x^2 + 2.487673537x + 3.430571965$$

For this particular version of least square fitting, the geometric interpretation of the error function E is the square of the Euclidean distance between $a\vec{x}_3 + b\vec{x}_2 + c\vec{x}_1 + \vec{d}$ and the vector $\vec{y}$, where $\vec{x}_j = (x_1^j, x_2^j, \ldots, x_n^j)$, $\vec{d} = d(1, 1, \ldots, d)$ and $\vec{y} = (y_1, \ldots, y_n)$. The best fit is thus achieved when the coefficients a, b, c and d are chosen so that $a\vec{x}_3 + b\vec{x}_2 + c\vec{x}_1 + \vec{d}$ is the orthogonal projection of the vector $\vec{y}$ onto the subspace of $\mathbb{R}^n$ spanned by $\vec{x}_3$, $\vec{x}_2$, $\vec{x}_1$ and $(1, 1, \ldots, 1)$. That is to say, the best fit occurs when the vectors $\vec{y} - (a\vec{x}_3 + b\vec{x}_2 + c\vec{x}_1 + \vec{d})$ and $a\vec{x}_3 + b\vec{x}_2 + c\vec{x}_1 + \vec{d}$ are perpendicular. You should verify that this is the case for the solution given above.

In both cases, you should consider why there is only one critical point for the error function, and that is of necessity a global minimum.

2.3 Fitting other types of funtions

Note that all the previous constructions relied upon was that the type of function we try to fit depend linearly on the coefficients. The same scheme works equally well to fit functions such as $\sum_{i=1}^{n} \sin(nx)$, although note that here we need to solve n simultaneous linear equations.

Even if the function does not depend linearly on the coefficients, it can sometimes be transformed to one which does. For example, if our data points $\{(x_i, y_i)\}$ were believed to approximate an exponential function of the form $y = ae^{kx}$, then setting

$$E(a, k) = \sum_{i=1}^{m} (y_i - ae^{kx_i})^2 .$$

would require us to solve the system

$$\sum_{i=1}^{m}(y_i - ae^{kx_i})e^{kx_i} = 0 \quad \sum_{i=1}^{m}(y_i - ae^{kx_i})ax_ie^{kx_i} = 0\,.$$

While this isn't impossible, it is much more straightforward to make a change of variables.

Assuming the y_i are all positive (which is reasonable since we believe the data is exponential), we can let $z_i = \ln y_i$. Then we are trying to fit a function $z = \ln\left(ae^{kx}\right)$, which reduces to the line $z = \ln a + kx$. This is familiar territory, and we can just proceed as before.

2.4 Fitting a circle

In a slightly different vein, suppose that the data points $\{(x_i, y_i)\}$ lie near a circle of unknown center and radius. Note that if we did know the center, this problem would be very simple: the desired radius would be the average distance from the points to the center. If you don't think very hard, you might suspect that the desired center would be very near the center of mass of the points $\{(x_i, y_i)\}$, but this will only be the case if the points are evenly distributed around the circle.

Instead, we can come up with a function that measures the error between an arbitrary circle and our data points.

Recall that the general equation of a circle centered at (a, b) with a radius of r is

$$(x-a)^2 + (y-b)^2 = r^2.$$

Unlike the previous cases, there is no independent variable. (Note that we *could* try to fit $y_i \simeq b \pm \sqrt{r^2 - (x_i - a)^2}$, but not only would the resulting equations be messy, this would bias things very badly; do you see why?). Nevertheless, we press on.

One reasonable measure of the distance between the points and a circle is the "area difference", that is

$$E(a,b,r) = \sum_i \left((x_i - a)^2 + (y_i - b)^2 - r^2\right)^2.$$

One difficulty with this is that E is not quadratic in a, b, and r. However, Maple is able to solve the resulting equations anyway:

```
> cpts:=circle_pts():
  epsilon:=(pt,a,b,r) -> (pt[1]-a)^2 + (pt[2]-b)^2 -r^2;
```

$$\varepsilon := (pt,\ a,\ b,\ r) \to (pt_1 - a)^2 + (pt_2 - b)^2 - r^2$$

```
> E:= (a,b,r) ->   sum( epsilon(cpts[i],a,b,r)^2,  i=1..nops(cpts));
```

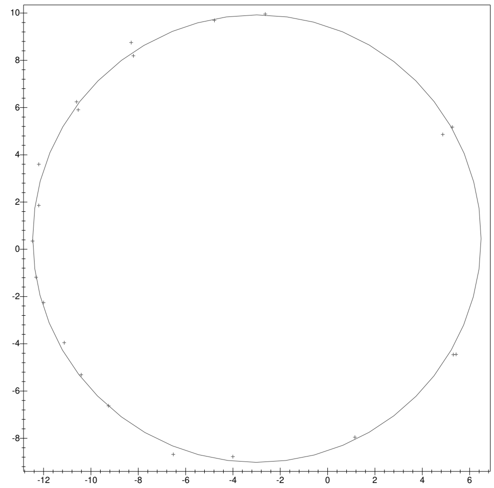
$$E := (a,\ b,\ r) \to \sum_{i=1}^{\mathrm{nops}(cpts)} \varepsilon(cpts_i,\ a,\ b,\ r)^2$$

```
> sol:= solve({diff(E(a,b,r),a)=0,
                diff(E(a,b,r),b)=0,
                diff(E(a,b,r),r)=0},
               {a,b,r});
```

$$
\begin{aligned}
sol := \quad & \{a = -7.279165053 - 12.95193570\, I,\ b = -2.210665541 + 2.912161761\, I,\ r = 0\},\\
& \{a = -7.279165053 + 12.95193570\, I,\ b = -2.210665541 - 2.912161761\, I,\ r = 0\},\\
& \{b = .4328495120,\ a = -4.481107455,\ r = 0\},\\
& \{b = -14.49583861 - 13.00051962\, I,\ a = 4.676168966 - 18.08449394\, I,\ r = 0\},\\
& \{a = 4.676168966 + 18.08449394\, I,\ b = -14.49583861 + 13.00051962\, I,\ r = 0\},\\
& \{r = 9.471322781,\ a = -2.987792224,\ b = .4467421770\},\\
& \{r = -9.471322781,\ a = -2.987792224,\ b = .4467421770\}
\end{aligned}
$$

Here we find a number of critical points for the function E, but from physical considerations, the only reasonable choice is the circle with center at $(-2.987792224, .4467421770)$ and radius 9.471322781. In order to see the fit, we plot the points and the circle. Note that we represent the circle parametrically, and use `subs` to substitute the desired solution.

```
> pp:=plot(cpts,style=point,scaling=constrained,axes=boxed):
  cp:=plot(subs(sol[6],[a+r*cos(t),b+r*sin(t),t=0..2*Pi])):
  plots[display](pp,cp);
```



The fact that there is only one interesting critical point is no accident, however. If we let $k = a^2 + b^2 - r^2$, then the resulting error function $H(a, b, k) = E(a, b, \sqrt{a^2 + b^2 - k})$ is quadratic in a, b, and k, and so we really only need to solve a linear system. It is easy to check that this new functional H still satifies all the criteria we wanted (that is $H(a, b, k) = 0$ if and only if all the data points lie on the circle $C(a, b, k)$, H is non-negative, and it is smooth), so fitting a circle becomes a linear problem after all. The reader should verify that, in fact, the unique minimum of H corresponds exactly to the critical points the original function E for which $r \neq 0$.

The interested reader might want to consider a similar approach to fitting other conic sections, such as an ellipse or a hyperbola, to given data. Do you expect to be able to make the problem linear, as in the case of the circle?

Another interesting variation on least-squares fitting is if we consider the measurement errors to occur in both x_i and y_i. In this case, we want to minimize the shortest distance to a line $y = mx + b$ rather than the vertical distance. In this case, the resulting equations will not be linear, nor can they be made linear. However, maple will be able to find the critical points with no trouble. There will always be at least two (there may be a third with a huge slope and intercept)– one is the minimum, and the other is a saddle point. It is worth thinking for a few minutes about the geometric interpretation of the saddle point in terms of the problem at hand.

2.5 Robust fitting

A discussion needs to be inserted about fitting lines using

$$E(m,b) = \sum_{i=1}^{n} ln\left(1 + (mx_i + b - y_i)^2\right) \, .$$

and

$$E(m,b) = \sum_{i=1}^{n} |mx_i + b - y_i| \, .$$

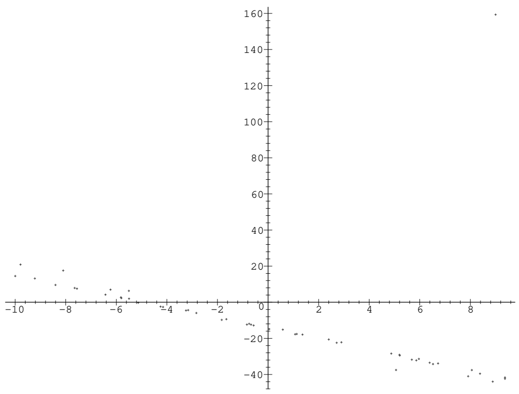
The basic idea is that there is nothing truly magical about minimizing sums of squares, although it is the appropriate thing to do if the errors are Gaussian (see next section). But note that if one has data with large errors ϵ_i, these will have a huge influence (because we use ϵ_i^2). The two mentioned above grow linearly (or sub-linearly) with $|\epsilon_i|$, causing the tails to count much less heavily.

```
> read(`lsq_data.txt`);
```

```
defined line\_pts(), bad\_line\_pts(), quadratic\_pts(), cubic\_pts(),
and circle\_pts()
```

First, we generate some "noisy" data.

```
> pts:=bad_line_pts():
> plot(pts,style=point);
```



```
> epsilon:= (pt,m,b) -> (m*pt[1]+b-pt[2]);
```

$$\varepsilon := (pt,\, m,\, b) \to m\, pt_1 + b - pt_2$$

First, let's find the line given by least squares:

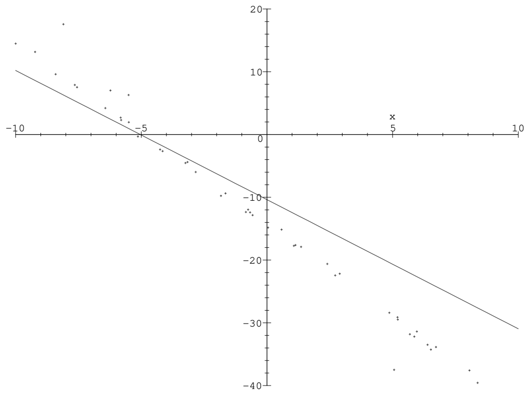
```
> H:=(m,b)->sum(epsilon(pts[i],m,b)^2,i=1..nops(pts));
```

$$H := (m,\, b) \to \sum_{i=1}^{\mathrm{nops}(pts)} \varepsilon(pts_i,\, m,\, b)^2$$

```
> sol:=solve(\{diff(H(m,b),m)=0, diff(H(m,b),b)=0\},\{m,b\});
```

$$sol := \{m = -2.058151695,\, b = -10.36674445\}$$

```
> p := plot(pts,style=point):
  l := plot(subs(sol,m*x+b),x=-10..10):
  plots[display](p,l,view=-40..20);
```



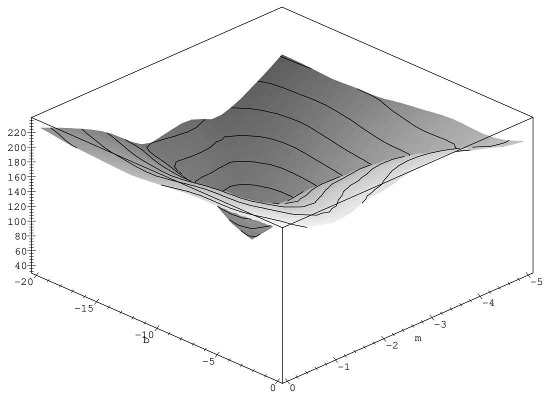
Now, let's try it minimizing $\ln(1+\epsilon_i^2)/2$, which behaves like ϵ^2 for small errors, but grows very slowly for large ones.

```
> R:=(m,b)->sum(ln(1+epsilon(pts[i],m,b)^2/2),i=1..nops(pts));
```

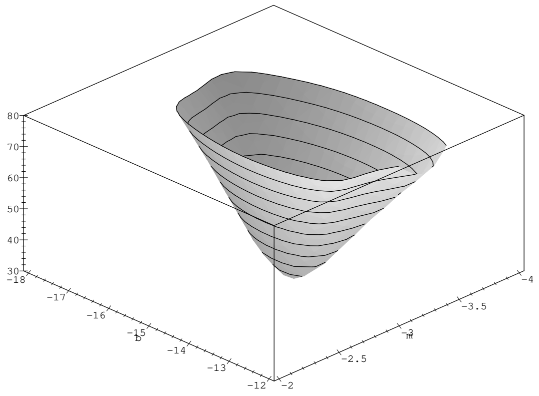
$$R := (m,\, b) \to \sum_{i=1}^{\mathrm{nops}(pts)} \ln(1 + \frac{1}{2}\,\varepsilon(pts_i,\, m,\, b)^2)$$

Here's what the functional we want to minimize looks like:

```
> plot3d(R(m,b),m=-5..0,b=-20..0,style=patchcontour, axes=boxed);
```



```
> plot3d(R(m,b),m=-4..-2,b=-18..-12,view=30..80, style=patchcontour, axes=boxed);
```



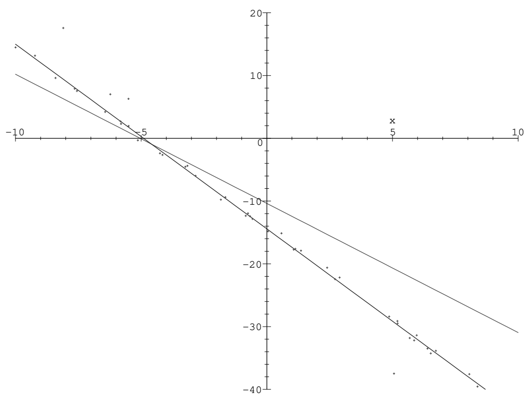
The equations we want to solve are not linear. In fact, they're messy enough that maple needs some help to find the minimum. From the pictures above, we can choose an appropriate region.

```
> rs:=fsolve( \{diff(R(m,b),m)=0, diff(R(m,b),b)=0\},
              \{m,b\},m=-3.1..-2.8, b=-15..-14);
```

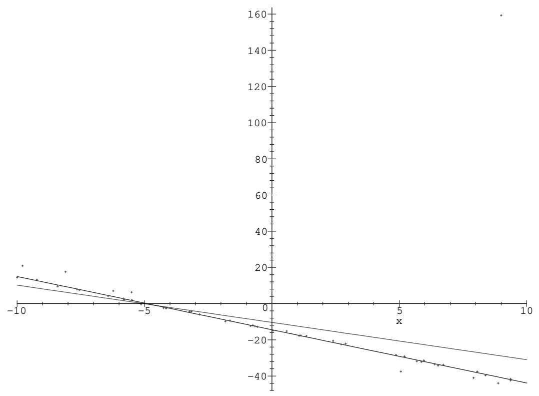
$$rs := \{b = -14.41947456, m = -2.942250792\}$$

And here we can compare the two lines found.

```
> p := plot(pts,style=point):
  l := plot(subs(sol,m*x+b),x=-10..10):
  s := plot(subs(rs,m*x+b),x=-10..10,color=blue):
  plots[display](p,l,s,view=-40..20);
```




```
> plots[display](p,l,s);
```



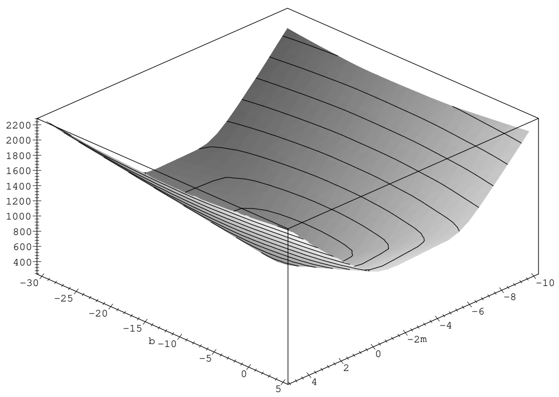
We can also try to minimize $\sum |\epsilon_i|$. Of course, this isn't even differentiable near $\epsilon_i = 0$, so we have to work harder to get less.

```
>   A:=(m,b)->sum(abs(epsilon(pts[i],m,b)),i=1..nops(pts));
```

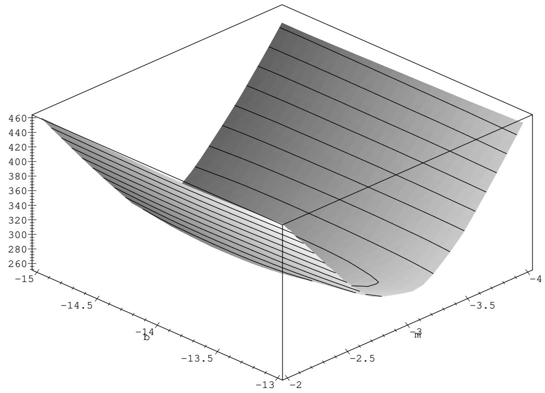
$$A := (m, b) \rightarrow \sum_{i=1}^{\mathrm{nops}(pts)} |\varepsilon(pts_i, m, b)|$$

We can determine graphically about where the minimum is, but to coerce maple into finding the minimum, we would have to work rather hard. There *are* reliable and efficient ways to determine the location of the minimum to any precision (at least given some hints), but they inappropriate for consideration here.

```
>   plot3d(A(m,b),m=-10..5,b=-30..5,style=patchcontour, axes=boxed);
```




```
>   plot3d(A(m,b),m=-4..-2,b=-15..-13,style=patchcontour, axes=boxed);
```



Repeatedly narrowing in on the minimum graphically will locate the choice of m and b.

2.6 A nod toward statistics

Here we really need a more complete discussion about *why* the sums of squares is what is usually mimimized (there are other considerations besides the fact that it gives a set of linear equations), and that we need to account for the variance of the ϵ_i. Further, we need to discuss correlation and goodness of fit (maybe with a mention of the incomplete gamma function?), possibly the normal distribution, etc. In particular, it is worth noting that the sums of the squares comes out of taking the log of the product of the errors, if errors are normal, and so on.