Chapter 2

If the Curve Fits, Wear It

It is not uncommon in science and mathematics to have a collection of data points

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

for which we believe there is a functional relationship between the x_i and the y_i. Often, we want to get some idea of what might be happening at points other than those for which we have data. That is, we want to find some f so that $f(x_i) = y_i$ for each of our points. Of course, in order to do this, we will need to have some knowledge (or make some guesses) about the function f, or sometimes about how precisely the points are known. In this chapter, we'll look at several different approaches to this problem of fitting a curve through (or near) some given data.

1 Interpolation

First, we will look at the situation where we the data points are assumed to be known exactly (or at least to good enough precision that we ignore any errors), and we want to find a curve of some type is chosen that passes through each of the data points. This practice is called *interpolation*. In the latter part of this chapter (beginning with §2), we will relax the restriction that the curve pass exactly through the points.

Of course, there are serious restrictions involved in interpolation. Since a line is determined by two points, if we have more points, we must either use a higher degree polynomial, or use several line segments. If we have n data points we wish to interpolate, either we can fit a polynomial of degree $n-1$ or find $n-1$ line segments which "connect the dots". We'll address both of these in this section, as well as a method which is a cross between the two approaches.

1.1 Polynomial Interpolation

Given n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, there is a unique polynomial of degree $n-1$ passing through them. Finding this polynomial is just a matter of solving the n linear equations

for the coefficients. For example, suppose we have 4 data points (x_1, y_1), (x_2, y_2), (x_3, y_3), and (x_4, y_4). There is exactly one degree 3 polynomial $p(x) = ax^3 + bx^2 + cx + d$ which passes through all four points. Plugging in our data yields the equations

$$\begin{aligned} ax_1^3 + bx_1^2 + cx_1 + d &= y_1 \\ ax_2^3 + bx_2^2 + cx_2 + d &= y_2 \\ ax_3^3 + bx_3^2 + cx_3 + d &= y_3 \\ ax_4^3 + bx_4^2 + cx_4 + d &= y_4 \end{aligned}$$

These are easily solved for the coefficients a, b, c, and d.

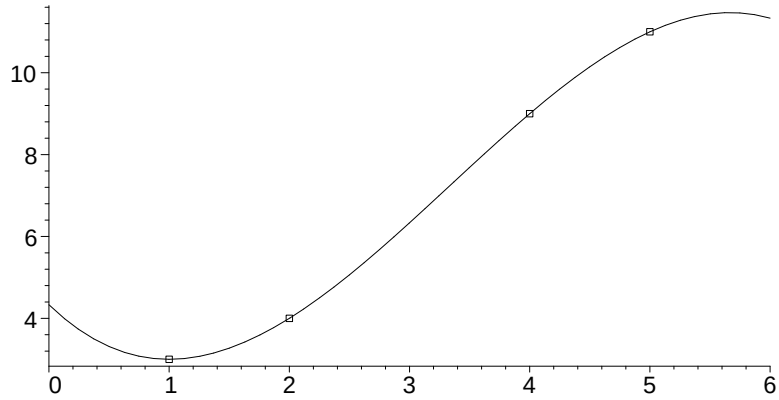
Maple 7[1] has a built-in command called `PolynomialInterpolation` to do this automatically for us, as part of the `CurveFitting` package. We'll do a brief example.

```
> data:=[[1,3],[2,4],[4,9],[5,11]]:

  with(CurveFitting):
  cubfit:=PolynomialInterpolation(data,x);
```

$$cubfit := -\frac{1}{6}x^3 + \frac{5}{3}x^2 - \frac{17}{6}x + \frac{13}{3}$$

```
> plots[display](
    plot(data,style=point,color=black,symbol=BOX),
    plot(cubfit,x=0..6,thickness=2));
```



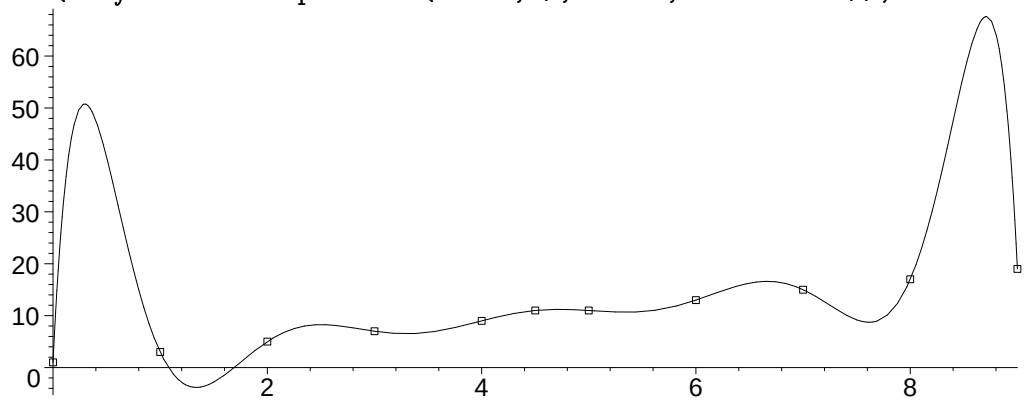
That seems to have worked pretty well. However, we should remark that polynomials are somewhat inflexible: minor variations in the data can have drastic effects on the behavior between the points. To emphasize this, we'll generate 10 points along the line $y = 2x + 1$, with one more point in the middle that is a bit above the line.

```
> data2:=[seq([i,2*i+1],i=0..4),[9/2,11],seq([i,2*i+1],i=5..9)];
```

[1] earlier versions of Maple have a similar command called `interp` with slightly different syntax.

$$data2 := [[0,\ 1],\ [1,\ 3],\ [2,\ 5],\ [3,\ 7],\ [4,\ 9],\ [\frac{9}{2},\ 11],\ [5,\ 11],\ [6,\ 13],\ [7,\ 15],\ [8,\ 17],\ [9,\ 19]]$$

```
> plots[display](
    plot(data2,style=point,color=black,symbol=BOX),
    plot(PolynomialInterpolation(data2,x),x=0..9,thickness=2));
```



This doesn't look much like a straight line, does it? Even though there is only one data point which is 1 unit above the line $y = 2x + 1$, at $x = 1/2$ the polynomial is about 50 units above this line.

1.2 Connect-the-Dots and Splines

As we said at the start of this section, another choice is not to use a single function, but to use several segments which connect the dots. The result would be a piecewise-affine function consisting of several line segments defined on a number of intervals. In the example using `data2` from the previous section, this function would be

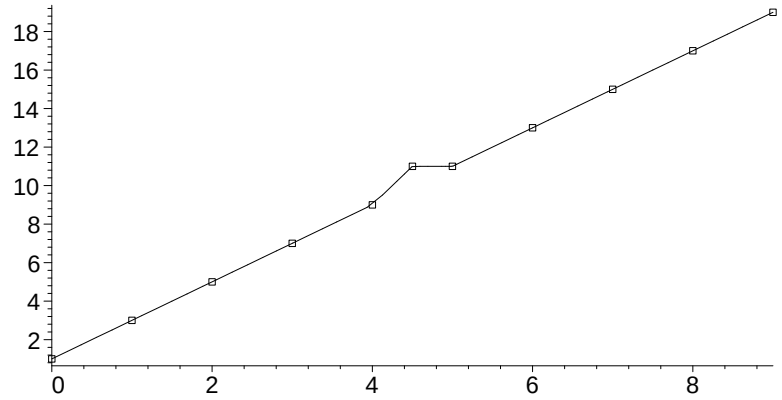
$$f(x) = \begin{cases} 2x+1 & \text{if } x \le 4 \\ 4x-7 & \text{if } 4 < x \le \frac{9}{2} \\ 11 & \text{if } \frac{9}{2} < x \le 6 \\ 2x+1 & \text{if } 6 < x \end{cases}$$

Although writing a maple procedure to figure this out for us would not be hard, such a piecewise affine curve is a version of something called a spline, which we will discuss more soon. This type of connect-the-dots curve is a linear (or degree 1) spline, and Maple can find it using the `Spline` command out of the `CurveFitting`[2] package.

```
> lspline:=Spline(data2,x,degree=1):
```

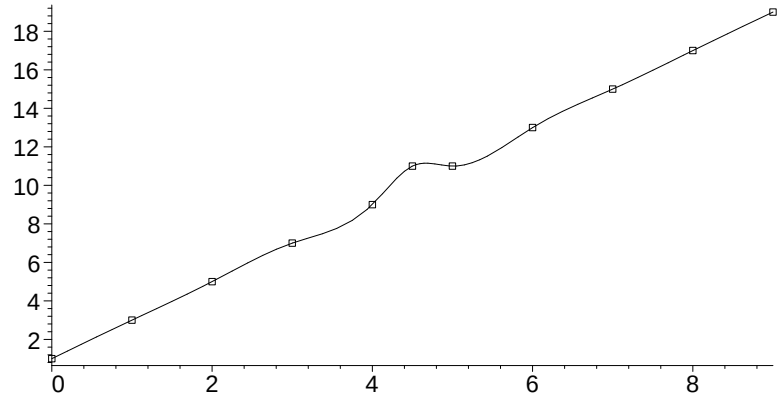
[2] versions prior to Maple 7 have a similar command called `spline` which takes slightly different arguments.

```
> plots[display](
    plot(data2,style=point,color=black,symbol=BOX),
    plot(lspline,x=0..9,thickness=2));
```



If we leave off the `degree=1`, Maple gives us a cubic spline instead:

```
> plots[display](
    plot(data2,style=point,color=black,symbol=BOX),
    plot(Spline(data2,x),x=0..9,thickness=2));
```



Notice how the cubic spline fits the points about as well as the "connect-the-dots" piecewise affine curve, but has no corners. What is this curve that Maple is showing us?

To make a spline, instead of connecting each pair of points by a straight segment, we put in a polynomial of degree d. We must ensure that the polynomial passes through the two endpoints of each segment, but this still leaves us an additional $d-1$ conditions per polynomial segment. The entire curve can be made smoother by choosing each polynomial segment so that the first $d-1$ derivatives at one endpoint agree with the derivatives of the previous segment. Since there is no segment before the first or after the last, this leaves $d-1$ more conditions still to specify. For a "natural spline", the high order derivatives at the endpoints are set to zero:

$$p^{(d-1)}(x_1) = p^{(d-2)}(x_1) = \ldots = p^{(\frac{d-1}{2})}(x_1) = 0$$

and

$$p^{(d-1)}(x_n) = p^{(d-2)}(x_n) = \ldots = p^{(\frac{d-1}{2})}(x_n) = 0$$

(if $d-1$ is odd, we have one fewer zero derivative at x_n than at x_1). Thus, for a cubic spline, the second derivatives vanish at eachpoint, and the polynomial segments have the same value, slope, and concavity at each interior point.

You might think that using higher degree polynomials in between would give a better appearance, but the same issues that arose in earlier begin to show up. Note that if we have n data points, fitting a spline of degree $n-1$ is exactly the same as the interpolating polynomial of degree $n-1$. Cubic splines are the most widely used, because they look quite smooth to the eye, but have enough freedom that they don't have to wiggle too much in order to pass through each data point. Many computer drawing programs have the ability to fit cubic splines through a set of points.

2 When the data is approximate

Now we relax the restriction that the function we are searching for must pass exactly through each of the data points. This is the typical situation in science, where we make a measurement or do an experiment to gather our y values from the input x values.

More specifically, let's 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 parameters $c_1,\ldots,c_k$ 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

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

where x_{ij} and y_i are the measurements of x_j and y in the i^{th} experiment. These experimental results gives us information about the unknown coefficients c_i.

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 $(f_{c_1,\ldots,c_k}(x_1, x_2, \ldots, x_m) - y)^2$. 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 $c_1, \ldots, c_k$.

We should remark that in some cases, we might want to use $E(c_1, \ldots, c_k)/n$, the *mean-squared error*. This will not change the answer we get (because the minimum occurs at the same value of c), but does allow us to compare sets of data with different numbers of data points.

3 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). Our problem is to find the values of m and b will best fit this data. Or to put it simply and geometrically, what is the straight line that best fit the information contained in the data (x_1, y_1), ..., (x_n, y_n)?.

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

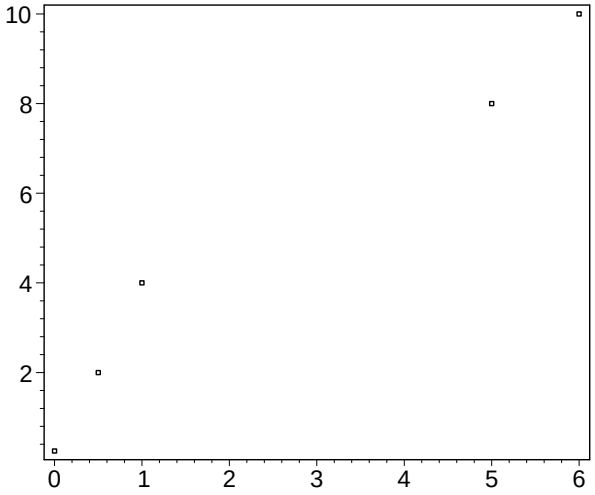
$$E(m, b) = \sum_{i=1}^{n} (mx_i + b - y_i)^2 \,.$$

The problem then becomes that of 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.

We can use the `LeastSquares` command from the `CurveFitting` package[3]

```
> CurveFitting[LeastSquares](pts,x);
```

$$1.271370968 + 1.431451613x$$

This says that the best values of m and b are $m = 1.431451613$ and $b = 1.271370968$, respectively. Now let's go through the steps ourselves, to better understand the process.

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

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

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

[3] As we noted before, the `CurveFitting` package was introduced in Maple 7. Earlier versions of Maple (as well as Maple 7) can accomplish the same thing using the `fit` command from the `stats` package, as follows.

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

As you can see, the syntax is a little different.

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

$$19.5625$$

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,pts);
```

$$3.125000000$$

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,pts), 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,pts), m)=0, diff( E(m,b,pts), 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 built-in version.

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.[4]

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

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

```
> m;
```

$$1.431451613$$

[4] 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 this operator instead.

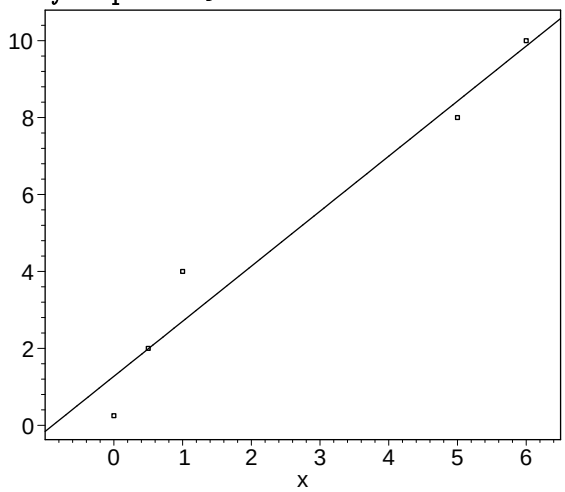
```
> E(m,b,pts);
```

$$2.929334677$$

The value of $E(m, b, pts)$ 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):
  display({plot(m*x+b, x=-1..6.5, axes=boxed),
           plot(pts, style=point)}):
```



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 resulting 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, and the discrepancy above is attributable to round-off errors..

As a final remark, we observe that if you now execute the command `diff( E(m,b,pts),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 $E(m, b, pts)$ for general m and b, you will have to unassign the values of m and b first:

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

4 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 fitting package.

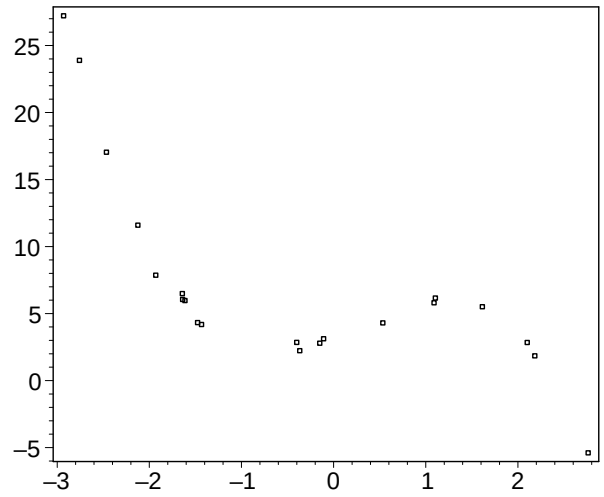
We randomly generate a set of 21 data points to be fitted to a cubic using a Maple program. (Don't worry about how this program works just now. We'll get into those details later).

```
> 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);
```



Now 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;
```

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

and then define the error function:

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

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

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

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

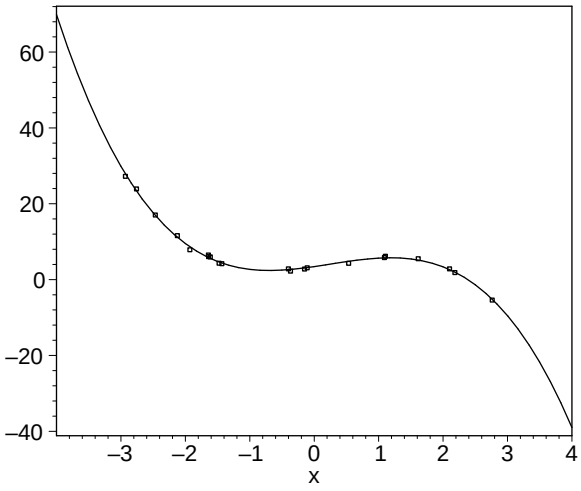
Of course, we do not see the answer, but the resulting values have already being assigned to the coefficients a, b, c, d:

```
> 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(CurveFitting):
  LeastSquares(data,x,curve=A*x^3+B*x^2+C*x+D);
```

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

For this particular version of least square fitting, we may once again interpret the error function E geometrically as 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 why it is of necessity a global minimum.

5 Fitting other types of funtions

In the previous constructions, we dealt with functions that depended linearly upon the parameters to be fitted. We found them by solving a linear system of equations, a relatively easy task.

However, as explained at the beginning of this chapter, the same scheme works equally

well for any function. In the general case, though, the resulting system that we must solve to find the best fit could depend non-linearly on the parameters, and the mere existence of solutions to such systems is not a trivial problem to settle. Take for instance, a function such as $y = \sum_{i=1}^{n} \sin(m_i x)$. Assuming measurements $(x_1, y_1), \ldots, (x_k, y_k)$, we may take as our error the function

$$E(m_1, \ldots, m_n) = \sum_{j=1}^{k} (y_j - \sum_{i=1}^{n} \sin(m_i x_j))^2 .$$

Its partial derivatives are quite easy to calculate, but it is far from clear if there are values of $m_1, \ldots, m_n$ where they all vanish simultaneously. For a particular set of values, we may ask Maple to solve the resulting system and get absolutely nothing. This either indicates an inability of Maple to handle such a system, or worse yet, the fact that such a system has no solution at all. Even when the latter happens, Maple itself might not be able to tell us so.

Sometimes, even if the function does not depend linearly on the coefficients, it can 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

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

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.

6 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.

We assume here that there is a routine called `circle_pts` which gives us points which lie near a circle of unknown radius and center. The one we use here is similar to that in §4, and is defined in `lsq_data.txt`.

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

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

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

$$\begin{aligned}
sol := [\quad & \{r = 0.,\ b = 8.598845417 - 5.760818468\,I,\ a = 10.74842676 - 3.044017219\,I\},\\
& \{r = 0.,\ a = 10.74842676 + 3.044017219\,I,\ b = 8.598845417 + 5.760818468\,I\},\\
& \{r = 0.,\ b = 9.123213697,\ a = 8.304601815\},\\
& \{r = 0.,\ a = 10.50158731 + 4.256560324\,I,\ b = 11.57538622 - 5.447431926\,I\},\\
& \{b = 11.57538622 + 5.447431926\,I,\ r = 0.,\ a = 10.50158731 - 4.256560324\,I\},\\
& \{a = 8.224446708,\ r = -4.468796472,\ b = 8.538288341\},\\
& \{a = 8.224446708,\ b = 8.538288341,\ r = 4.468796472\}]
\end{aligned}$$

Here we find a number of critical points for the function E, but from physical considerations, the only reasonable choice is the circle for which $r > 0$.

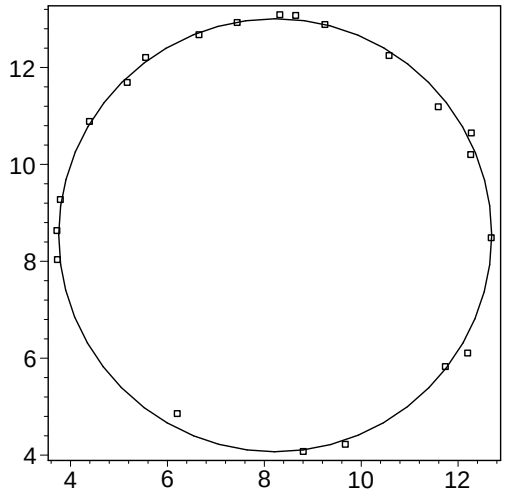
This is the seventh entry in the above list, so we could refer to it just by `sol[7]`. However, with a different set of data, it might be the second solution, or the fourth, etc. To ensure that we always get the right one, no matter what the order is, we can use Maple's `select` command to pull out the one with $r > 0$. This looks more daunting than it really is: we just define a function that returns `true` if that $r > 0$ for that solution, and `false` if not. Then `select` returns only those elements for which the function is true. We need to use `op`, because we have a list of sets, and we just want the one answer.

```
> goodsol:=op(select(s->if (subs(s,r)>0) then true else false fi,sol));
```

$$goodsol := \{a = 8.224446708,\ b = 8.538288341,\ r = 4.468796472\}$$

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.

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



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?

7 Robust fitting

Let us return momentarily to the problem discussed in §3: given some data $(x_1, y_1), \ldots, (x_n, y_n)$, we found the best line that fits it. This was accomplished by measuring the square of the *vertical* distance from each data point and the line $y = mx+b$, which led us to consider the error function

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

The problem was solved by finding the values of m and b where E achieves its minimum.

However, the square of the vertical distance is only one of many reasonable ways of measuring the distance from the data points to the line $mx + b$. For example, it is perfectly reasonable to consider instead the expression $1 + (mx_i + b - y_i)^2$. If (x_i, y_i) is on the line, this value would be equal to 1, and its logarithm would then be zero. Thus, the function

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

is also a reasonable way of measuring the distance from the data points to the line $y = mx + b$, and its minimum would produce a best fit line in this other sense. Similarly, We may argue that the function

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

is another reasonable alternative, though this time E is not even differentiable in the region of interest (if a data point (x_i, y_i) happens to be exactly on the line, $mx_i + b - y_i = 0$ and the absolute value is not differentiable at 0).

There is a priori no particular reason to prefer one error function over another. And for each choice of such we could end up with a problem whose solution exists and produces a line that best fit the data. Then we could compute the value of the error function for the best fit line, value that depends upon the choice of function made. A *canonical* way of choosing the best error function could be that for which this number is the smallest. But finding such a function is quite a difficult, if not impossible, problem to solve. If we limit our attention to *linear estimators*, in a sense to be explained in the last section of this chapter, the solution obtained in §3 is optimal. In here, we pursue further the problem of finding the best fit according to the two error functions introduced above.

Notice that if $\epsilon_i = mx_i + b - y_i$, data with large errors ϵ_i excert a huge influence on the original error function of §3 (because we use $\sum \epsilon_i^2$). The two error functions mentioned above

grow linearly (or sub-linearly) with $|\epsilon_i|$, causing the tails to count much less heavily. We will see this when comparing the different results.

In order to solve the problem now at hand, we first generate some "noisy" data. We load the routines in `lsq_data.txt`:

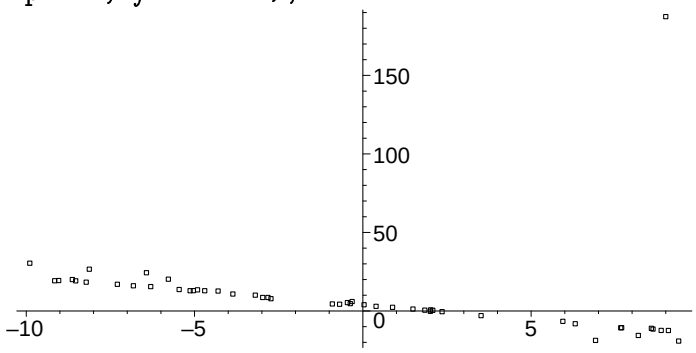
```
> read(`lsq_data.txt`);
```

```
defined line_pts(), bad_line_pts(), quadratic_pts(), cubic_pts(), and circle_pts()
```

The data and its graphical visualization can then be obtained by:

```
> pts:=bad_line_pts():
```

```
> plot(pts,style=point,symbol=box);
```



You should notice that while most of the data lies fairly near a line, there is one point which is extremely far away from the rest of the data.

In order to compare results, let us first find the line given by least squares. First, we define a function ϵ, which gives the signed vertical distance between a point `pt` and a line of slope `m` and intercept `b`

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

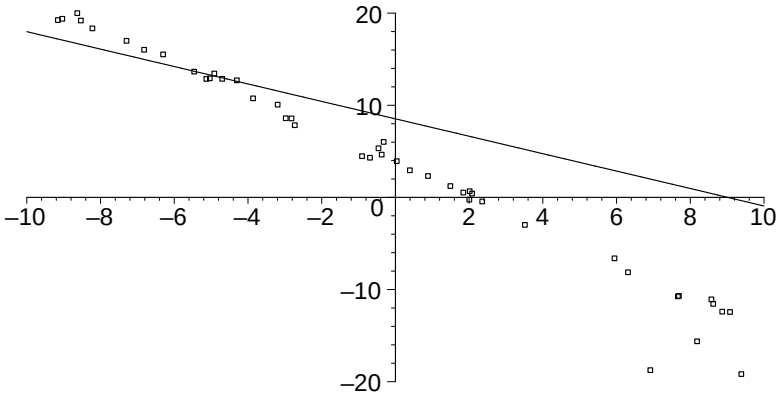
$$\varepsilon := (pt,\ m,\ b) \rightarrow m\, pt_1 + b - pt_2$$

Now we compute the regular least squares fit.

```
> H:=(m,b,pts)->sum(epsilon(pts[i],m,b))^2,i=1..nops(pts)):
  sol:=solve({diff(H(m,b,pts),m)=0, diff(H(m,b,pts),b)=0},{m,b});
```

$$sol := \{m = -.9455490512,\ b = 8.528125863\}$$

```
> display({plot(pts,style=point,symbol=box),plot(subs(sol,m*x+b),x=-10..10)},
          view=-20..20);
```



In the above, we restricted our attention to the region $-20 < y < 20$, where the line and the majority of the data lies. As you can see, the fit to the majority of the data is very poor, because of the influence exerted by the anomalous data point.

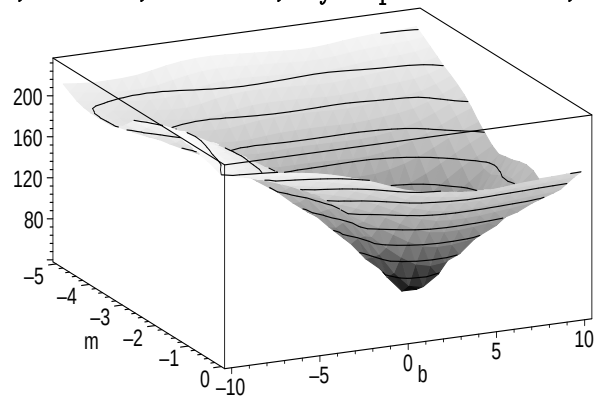
Now, let's try again, this time minimizing $\ln(1+\epsilon_i^2/2)$, which behaves like ϵ^2 for small errors, but grows very slowly for large ones.

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

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

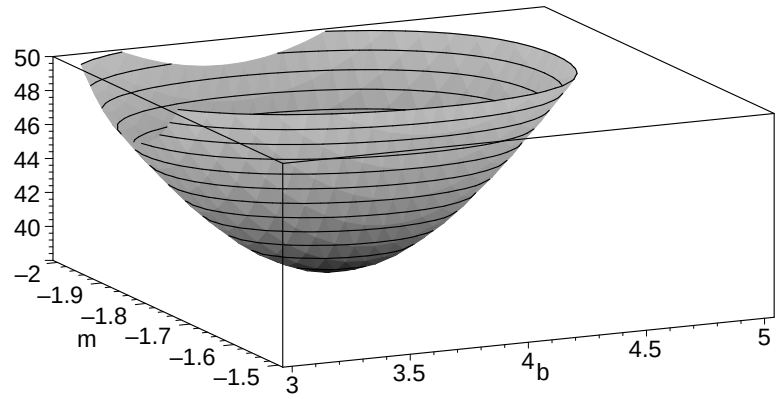
Here's what the functional we want to minimize looks like:

```
> plot3d(R(m,b,pts),m=-5..0,b=-20..0,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. By examining the plot and zooming in on the minimum, we can choose an appropriate region.

```
> plot3d(R(m,b,pts),m=-2..-1.5,b=3..5,view=38..50, axes=boxed);
```



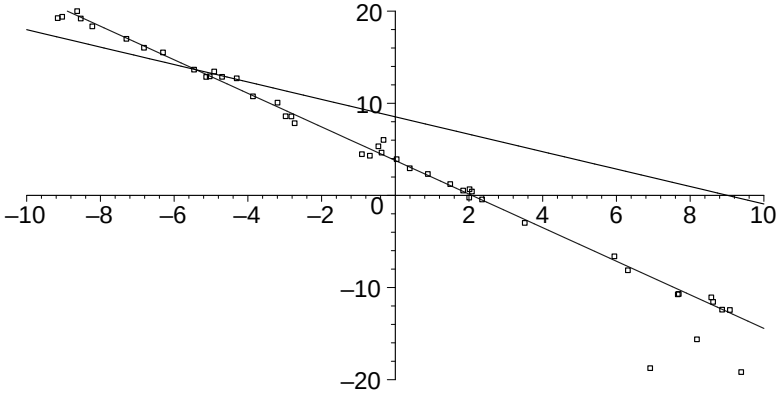
Once we have an appropriate region, we can ask Maple to look for the solution numerically using `fsolve`.

```
> rs:=fsolve( {diff(R(m,b,pts),m)=0, diff(R(m,b,pts),b)=0},
              {m,b},m=-2..-1.5, b=3..5);
```

$$rs := \{m = -1.822071903,\ b = 3.789519722\}$$

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=-20..20);
```



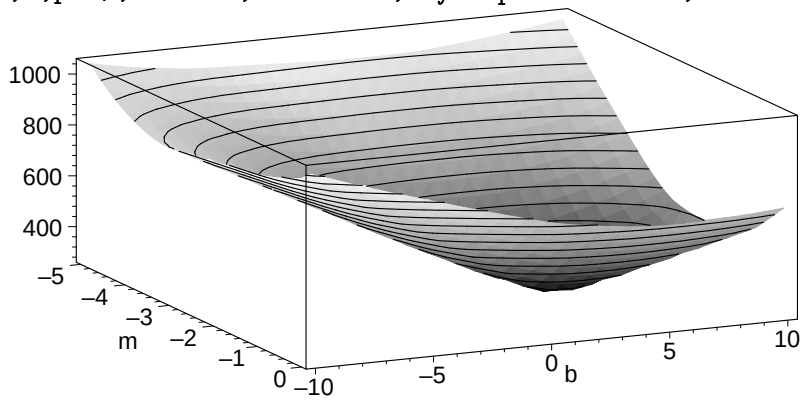
Let us now try to minimize $\sum |\epsilon_i|$. Since this function is not even differentiable near $\epsilon_i = 0$, we have to work harder to get less.

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

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

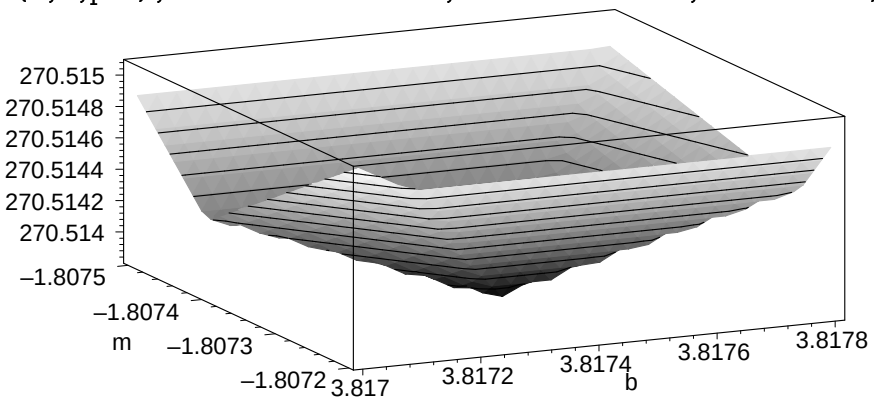
Displaying the graph of this function gives us an idea about where its minimum lies:

```
> plot3d(A(m,b,pts),m=-5..0,b=-10..10,style=patchcontour, axes=boxed);
```



But coercing Maple into finding a numerical approximation to it is not so easy. There *are* reliable and efficient ways to determine the location of the minimum to any precision (at least given some hints), but they are inappropriate for the scope of this chapter, and will not be considered here. Instead we may zero in on the minimum by looking at where the lowest point is and repeatedly restricting the domain. After several iterations, we get the following.

```
> plot3d(A(m,b,pts),m=-1.0875..-1.0872,b=3.817..3.8178, axes=boxed);
```



This leads us to a choice of $m = -1.80733$, $b = 3.8174$ as our "best" line. This is almost (but not quite) the same line found using our distance $R(m, b)$.

8 A nod toward statistics

In the problem we have treated so far, there have been a distinction made between the x and y coordinate of a data point (x_i, y_i): the x coordinate is thought as the predictor variable, while the y coordinate is the predicted value. The distinction is significant. If, for example, we think of x as a function of y, the best line that fits the data using the method of least square is, in general, different than the corresponding line when thinking of y as a function of x. Indeed, if we use the same data as in §3, the best line $x = ny + c$ that fits it is given by

$x = 0.6677953348y - 0.738807374$. Solving in terms of y yields $y = 1.497464789x + 1.106338028$; this differs significantly from the line $y = 1.431451613x + 1.271370968$ found in §3.

In this case, we use as error the square of the *horizontal* distance, and it is somewhat perturbing that this similarly reasonable approach leads to a best fit that differs from the one obtained when employing vertical distances.

The situation can be made even worse if neither x nor y is a predictor variable. In this case, we want to minimize the shortest distance to a line $y = mx + b$ rather than the vertical (or horizontal) distance. 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.

In practice, the perturbing fact that different but reasonable error functions lead to best fits that are different is resolved by knowing what part of the data is the input and what part is predicted from it. This happens often enough, though not always. We thus can make a good choice for E and solve a minimization problem. That settled, we are still left with the problem of demonstrating why our choice for E is a good one.

It is rather easy to write down the solution to the problem in §3: if

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i\,, \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i\,,$$

then

$$\hat{m} = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{j=1}^{n} x_j\right)\left(\sum_{j=1}^{n} y_j\right)}{n\sum_{i=1}^{n} x_i x_i + \left(\sum_{j=1}^{n} x_j\right)^2} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\,, \quad \hat{b} = \bar{y} - m\bar{x}\,.$$

If y_i are assumed to be the values of random variables Y_i which depend *linearly* upon the x_i,

$$Y_i = mx_i + b + \varepsilon_i\,,$$

with errors ε_i that are independent from each other, are zero on average and have some fixed variance, then the value of m given above is the *best* estimator of the slope among all those linear estimators that are unbiased. "Best" here is measured by calculating the deviation from the mean, and this best estimator is the one that produces the smallest such deviation.

This conclusion follows by merely making assumptions about the inner products of the data points $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$. Staticians often would like to answer questions such as the degree of accuracy of the estimated value of m and b. For that one would have to assume more about the probability distribution of the error variables Y_i. A typical situation is to assume that the ε_i above are normally distrubuted, with mean 0 and variance σ^2. Under these assumptions, the values of m and b given above are the so-called *maximum likelihood estimators* for these

two parameters, and there is yet one such estimator for the variance σ^2. But, since we assumed more, we can also say more. The estimators $\hat{m}$ and $\hat{b}$ are normally distributed and, for example, the mean of $\hat{m}$ is m and its variance is $\sigma^2 / \sum_{i=1}^{n}(x_i - \bar{x})^2$. With this knowledge, one may embark into determining the confidence we could have on our estimated value for the parameters. We do not do so in here, but want to plant the idea in the interested reader, whom we refer to books on the subject.