Feb 27, 2024.

A bit more on least squares and friends.

Proc to grab data from web

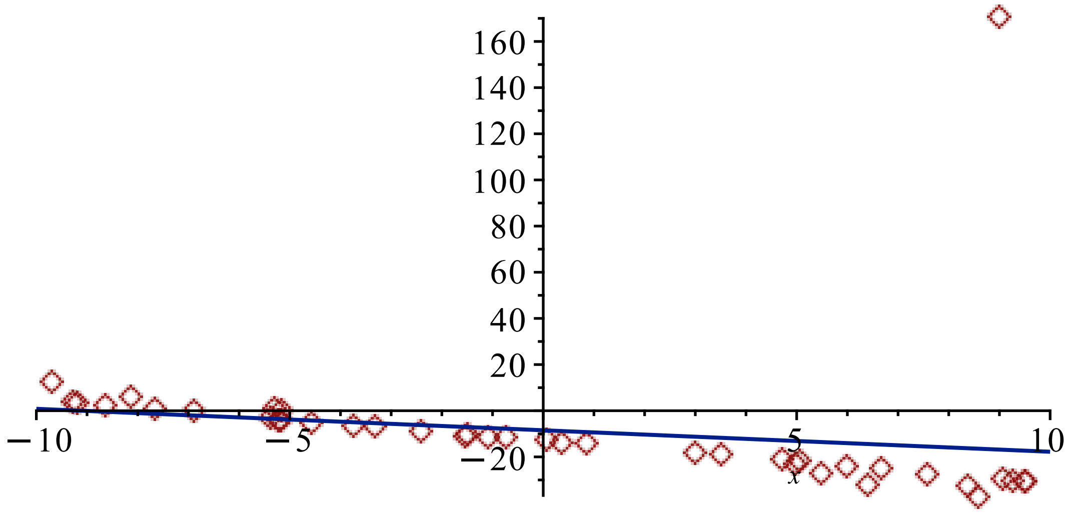
Let's pick up where we left off (ish).

```
> ExecuteFromWeb("https://www.math.stonybrook.edu/~scott/mat331.
  spr24/daily/extras/bad_line.txt")
loaded 40 points into lpts.
```

```
> with(CurveFitting):
  badfit:=LeastSquares(lpts,x);
```

$$badfit := -8.47217789980814 - 0.930101594702111\,x \tag{1}$$

```
> plot([lpts,badfit], x=-10..10, style=[point,line],symbolsize=20,
  size=[.8,.5]);
```



Let's do this by hand.
Assume m, b are unknowns.

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

$$sqErr := (pt, m, b) \mapsto \left(pt_2 - m \cdot pt_1 - b\right)^2 \tag{2}$$

```
> sqErr(lpts[2],m,b);
```

$$(-6.522580699 + 3.748081610\,m - b)^2 \tag{3}$$

```
> lsqDist:= (m,b,data) -> local i; add(sqErr(data[i],m,b),i=1..nops
  (data))/nops(data)
```

$$lsqDist := (m, b, data) \mapsto \frac{add\left(sqErr\left(data_i, m, b\right), i = 1..nops(data)\right)}{nops(data)} \tag{4}$$

```
> lsqDist(-0.93,-8.4,lpts);
```

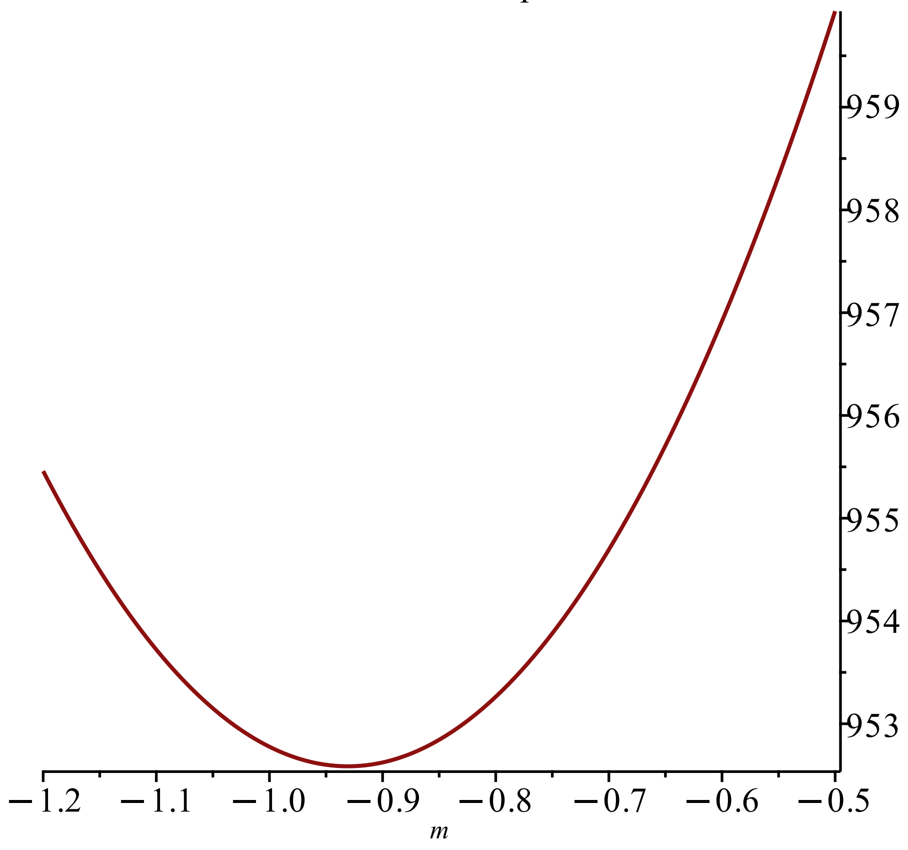
$$952.5861378 \tag{5}$$

```
> lsqDist(-0.95,-8.4,lpts);
```

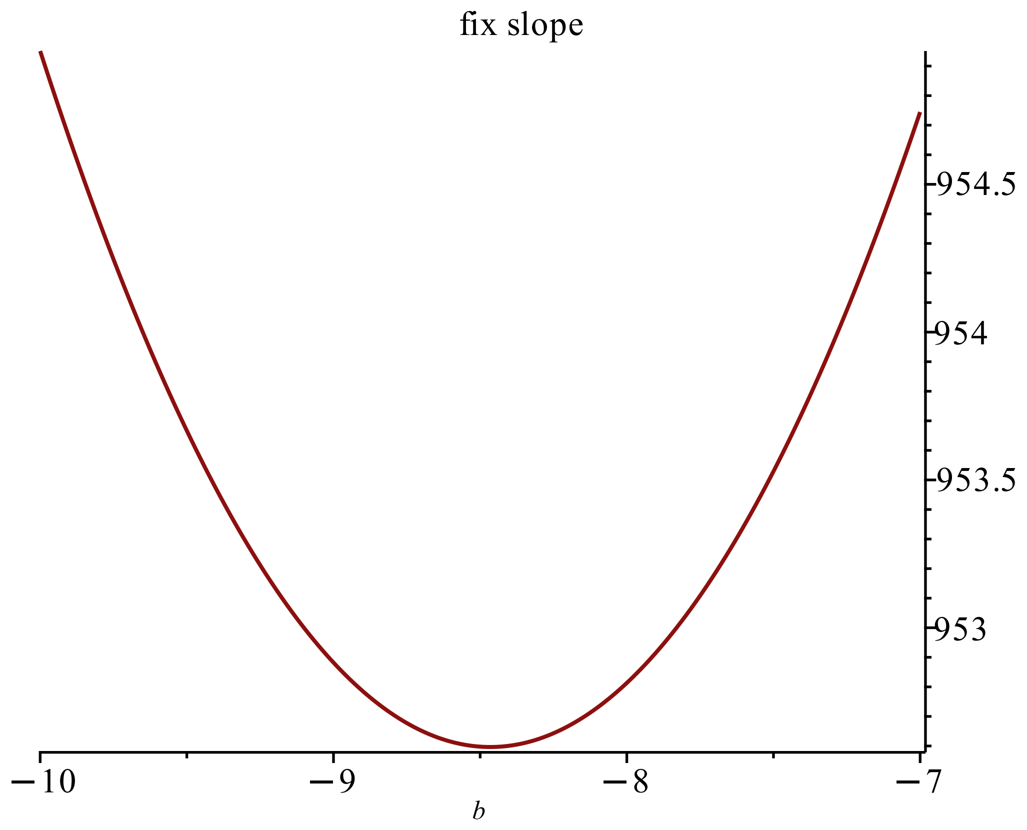
(6)

```
952.6008630                                                    (6)
> plot(lsqDist(m,-8.4,lpts),m=-1.2..-0.5, title="fix intercept");
```

fix intercept

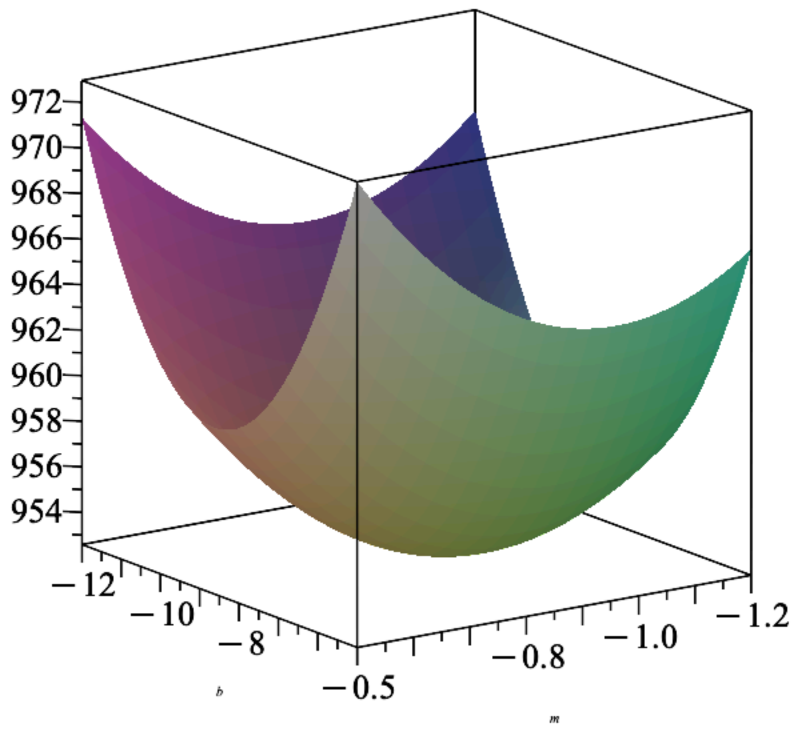


```
> plot(lsqDist(-0.95,b,lpts),b=-10..-7, title="fix slope");
```



```
> plot3d(lsqDist(m,b,lpts),m=-1.2..-0.5, b=-12..-5,title="minimize
  square", style=surfacecontour);
```

minimize square



We can clearly see the minimum of this function. Since Ldist is differentiable in the slope and intercept variables, we can solve for the critical point.

```
> Ldist:=lsqDist(m,b,lpts):
  diff(Ldist,m);
  diff(Ldist,b);
```

$$0.6625104446\, b + 79.21770381\, m + 79.29341901$$

$$2\, b + 0.6625104446\, m + 17.56055782 \qquad (7)$$

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

$$\{b = -8.472177899,\, m = -0.9301015950\} \qquad (8)$$

Note that we could try to minimize the absolute value of the distance...

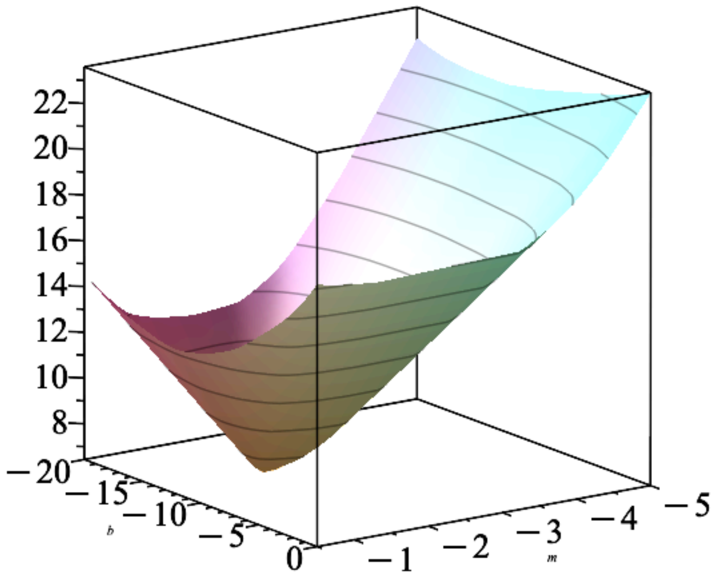
```
> absErr:= (pt,m,b) -> abs(pt[2] - (m*pt[1] + b));
  absDist:= (m,b,data) -> local i; add(absErr(data[i],m,b),i=1..nops
  (data))/nops(data)
```

$$absErr := (pt, m, b) \mapsto |pt_2 - m \cdot pt_1 - b|$$

$$absDist := (m, b, data) \mapsto \frac{add\left(absErr\left(data_i, m, b\right), i = 1 ..nops(data)\right)}{nops(data)} \qquad (9)$$

```
> plot3d(absDist(m,b,lpts),m=-5..-0.5, b=-20..0,title="minimize abs",
  style=surfacecontour);
```

minimize abs



But we can't take deriv of absolute value.

Let's look for a function that is quadratic near 0, but grows slowly for distances far from zero.

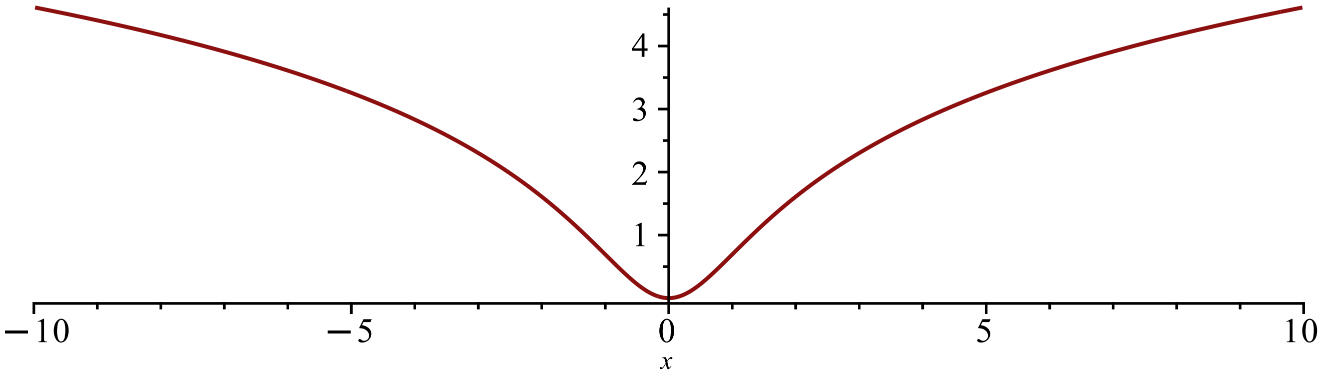
```
> series(ln(1+x),x)
```

$$x-\frac{1}{2}x^2+\frac{1}{3}x^3-\frac{1}{4}x^4+\frac{1}{5}x^5+\mathrm{O}(x^6) \qquad \textbf{(10)}$$

```
> series(ln(1+x^2),x)
```

$$x^2-\frac{1}{2}x^4+\mathrm{O}(x^6) \qquad \textbf{(11)}$$

```
> plot(ln(1+x^2),x=-10..10,scaling=constrained);
```



This seems a good candidate to weight the far-away points in a way that they don't screw things up, without ignoring them completly.

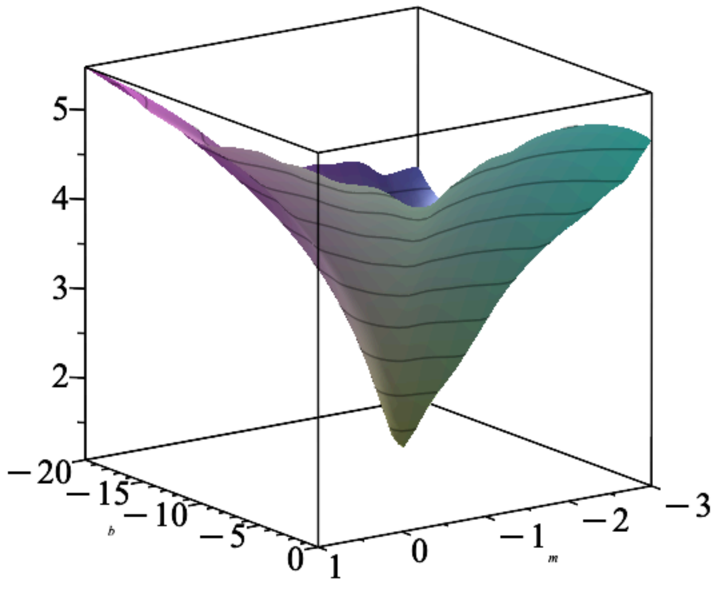
```
> logErr:= (pt,m,b) ->ln(1+(pt[2] - (m*pt[1] + b))^2);
  logDist:= (m,b,data) -> local i; add(logErr(data[i],m,b),i=1..nops
  (data))/nops(data)
```

$$logErr := (pt, m, b) \mapsto \ln\left(1 + \left(pt_2 - m \cdot pt_1 - b\right)^2\right)$$

$$logDist := (m, b, data) \mapsto \frac{add\left(logErr\left(data_i, m, b\right), i = 1 ..nops(data)\right)}{nops(data)} \qquad \textbf{(12)}$$

```
> plot3d(logDist(m,b,lpts),m=-3..1, b=-20..0,title="minimize log",
  style=surfacecontour);
```

minimize log



Sorry, I had some trouble solving for the derivatives equal to 0. I dunno, it usually works. But today is not my day.

So, instead, let's just use Maple's Minimize function, which is way faster anyway.
Maple has a built-in facility to search for minima of functions of one or more variables.
Minimizing such functions without using calculus has made siginificant progress in the last 20 years or so, and Maple has a pretty good procedure built it.

While you can use this in your project, note that it will miss the non-minimum critical point in part 3 (cuz it isn't a minimum), so you still need to do the calculus or the thinking to figure out what it means.

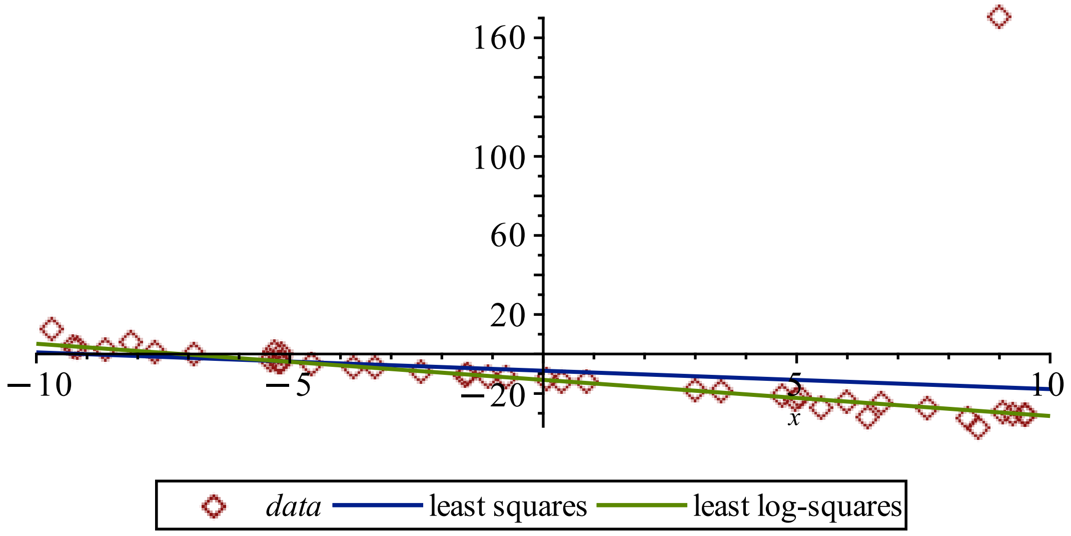
```
> Mini:=Optimization[Minimize](logDist(m,b,lpts), b=-20..0, m=-3..0)
```

$$Mini := [1.06636069982635995, [b = -13.0995320795104, m = -1.82808928324032]] \quad (13)$$

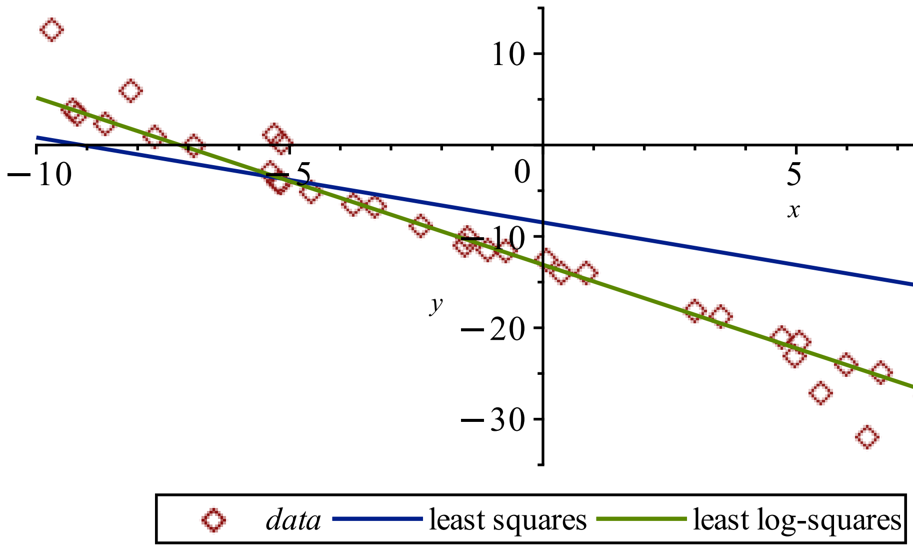
```
> MyLine:=subs(Mini[2],m*x+b)
```

$$MyLine := -1.82808928324032\,x - 13.0995320795104 \quad (14)$$

```
> plot([lpts,badfit,MyLine], x=-10..10, style=[point,line$2],
  symbolsize=20,size=[.8,.5], legend=[data,"least squares", "least
  log-squares"]);
```



```
> plot([lpts,badfit,MyLine], x=-10..10,y=-35..15, style=[point,
  line$2],symbolsize=20,size=[.8,.5], legend=[data,"least squares",
  "least log-squares"]);
```



Let't do another example, quickly, with even more bad points.

```
> ExecuteFromWeb("https://www.math.stonybrook.edu/~scott/mat331.
  spr24/daily/extras/worse_line.txt")
loaded 54 points into bdata.
```

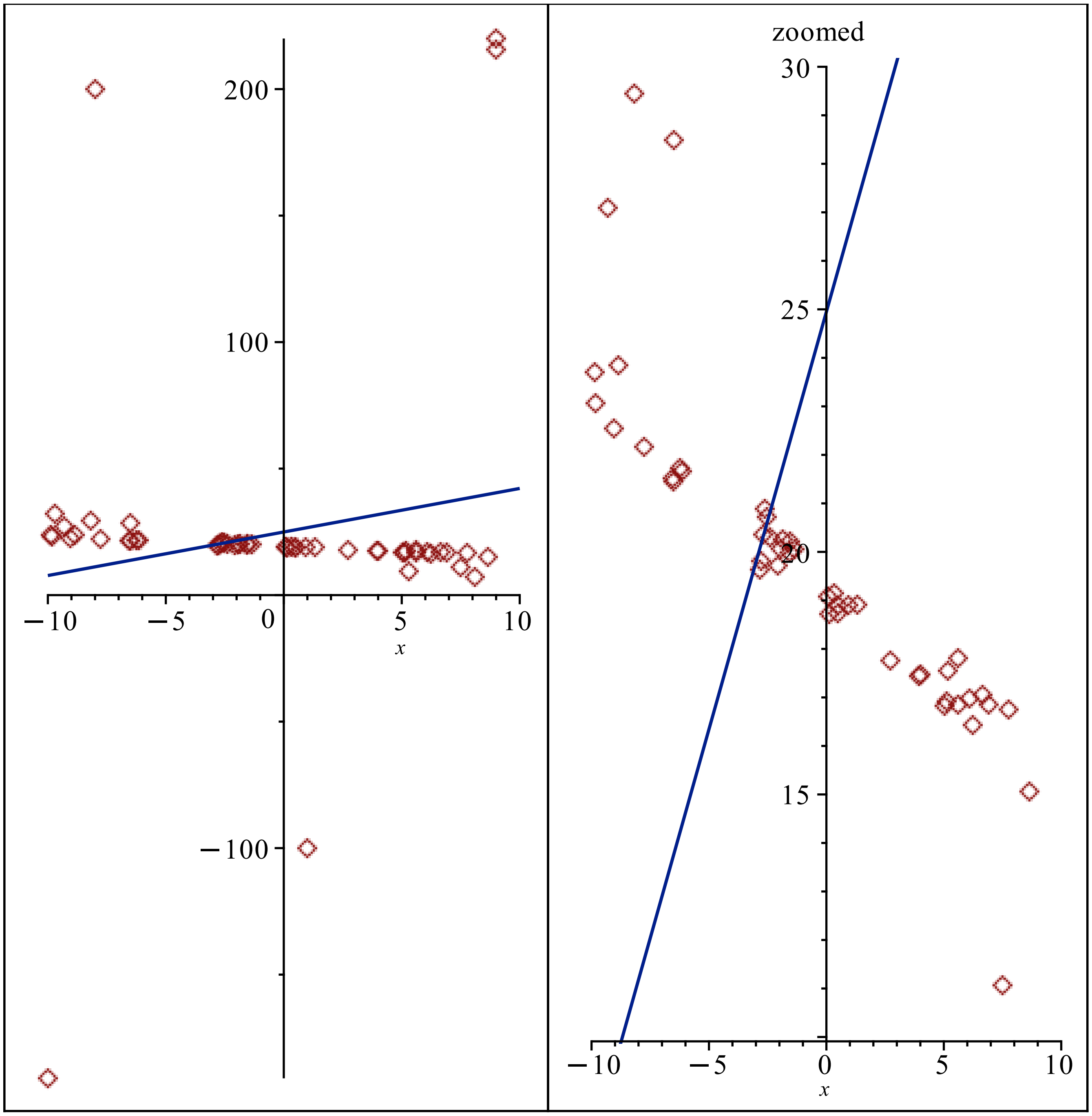
OK, we have 54 data points; let'ts fit them with least squares and see how it looks.

```
> bline := CurveFitting[LeastSquares](bdata, x)
```

$$bline := 24.9502982841362 + 1.72140920411782\,x \qquad (15)$$

This is pretty terrible:

```
> plots[display](<
     plot([bdata,bline], x=-10..10, style=[point,line], symbolsize=
  20) |
     plot([bdata,bline], x=-10..10,-10..10, style=[point,line],
  symbolsize=20, view=[-10..10,10..30],title="zoomed") >);
```



So now let's try to improve this using the least log-squares method. Clearly the line we are looking for has an intercept around 20, and a slope less than 0.

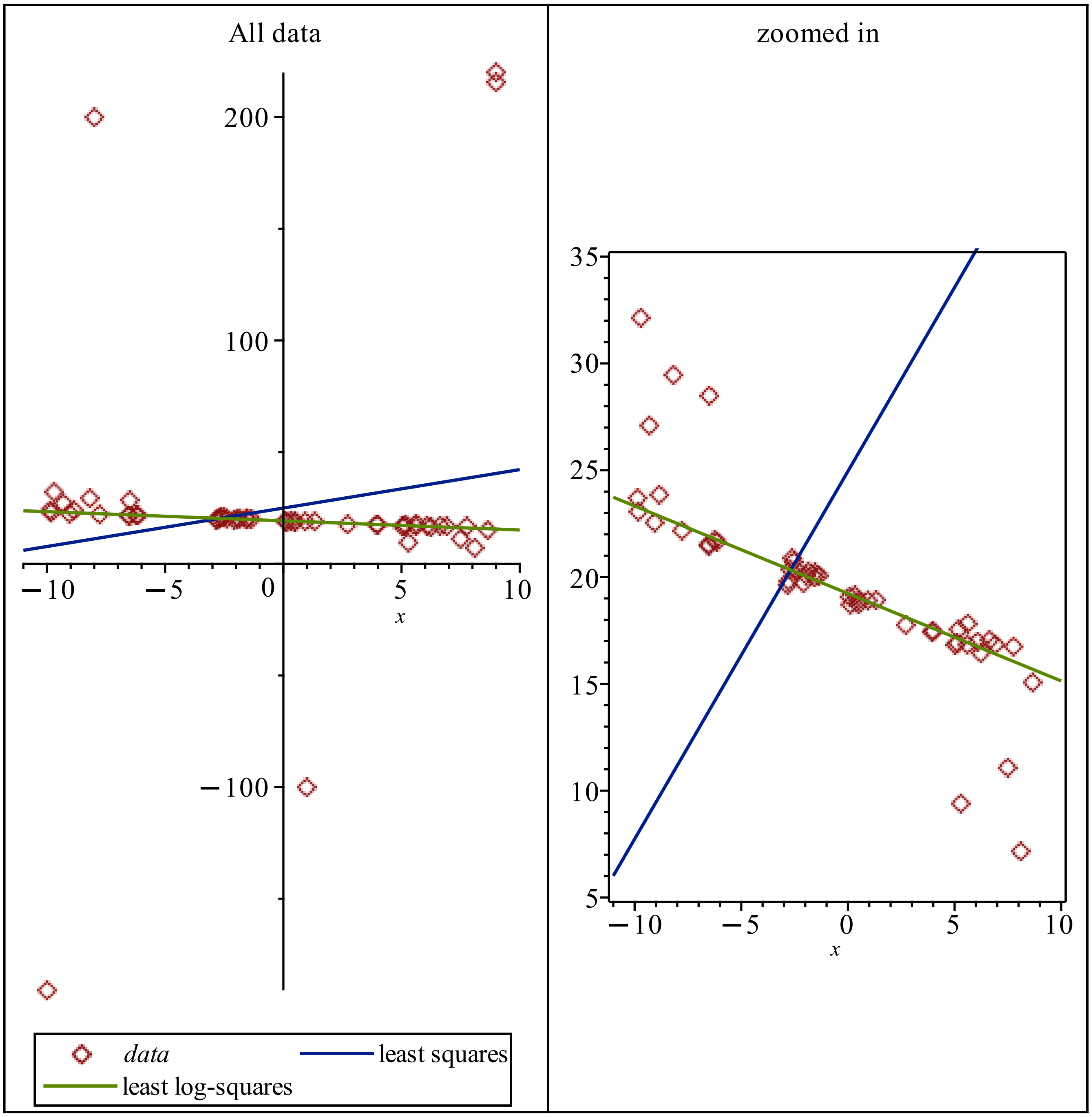
```
> TheMin:=Optimization[Minimize](logDist(m,b,bdata), b=-15..25, m=-5.
  .0)
```

$TheMin := [1.56085987773085733, [b = 19.2306735590914, m = -0.409601481190954]]$ **(16)**

```
> GoodLine:=subs(TheMin[2],m*x+b)
```

$GoodLine := -0.409601481190954\,x + 19.2306735590914$ **(17)**

```
> plots[display](
    <plot([bdata, bline, GoodLine], x = -11 .. 10, style = [point,
line $ 2], symbolsize = 19,
         title = "All data", legend = [data, "least squares",
"least log-squares"]) |
   plot([bdata, bline, GoodLine], x = -11 .. 10, style = [point,
line $ 2], symbolsize = 19, scaling=constrained,
         view = [-11 .. 10, 5 .. 35], axes = boxed, title = "zoomed
in")>);
```



The least-log-squares method is a "robust" fitting method, in that it is much less sensitive to data with some outliers (or big noise) thrown in than regular least squares, but acts like least squares for small variations.

There are other, more recent, methods that are also robust. In particular, the Least Trimmed Squares method (introduced in the 1980s, but significantly studied only in the last 20 years) examines all subsets throwing out k points, and picks the "best" line from among those subsets. This is a lot more computation that regular least squares, but still tractable now that computers are reasonably fast.

```
> trimline := subs(x[1] = x, Statistics[LeastTrimmedSquares](bdata,
  x))
```

$$trimline := -0.401442654921944\, x + 19.1216440573069 \quad (18)$$

(I don't know why maple insists on calling the variable x_1 instead of x, so I swapped it out.)

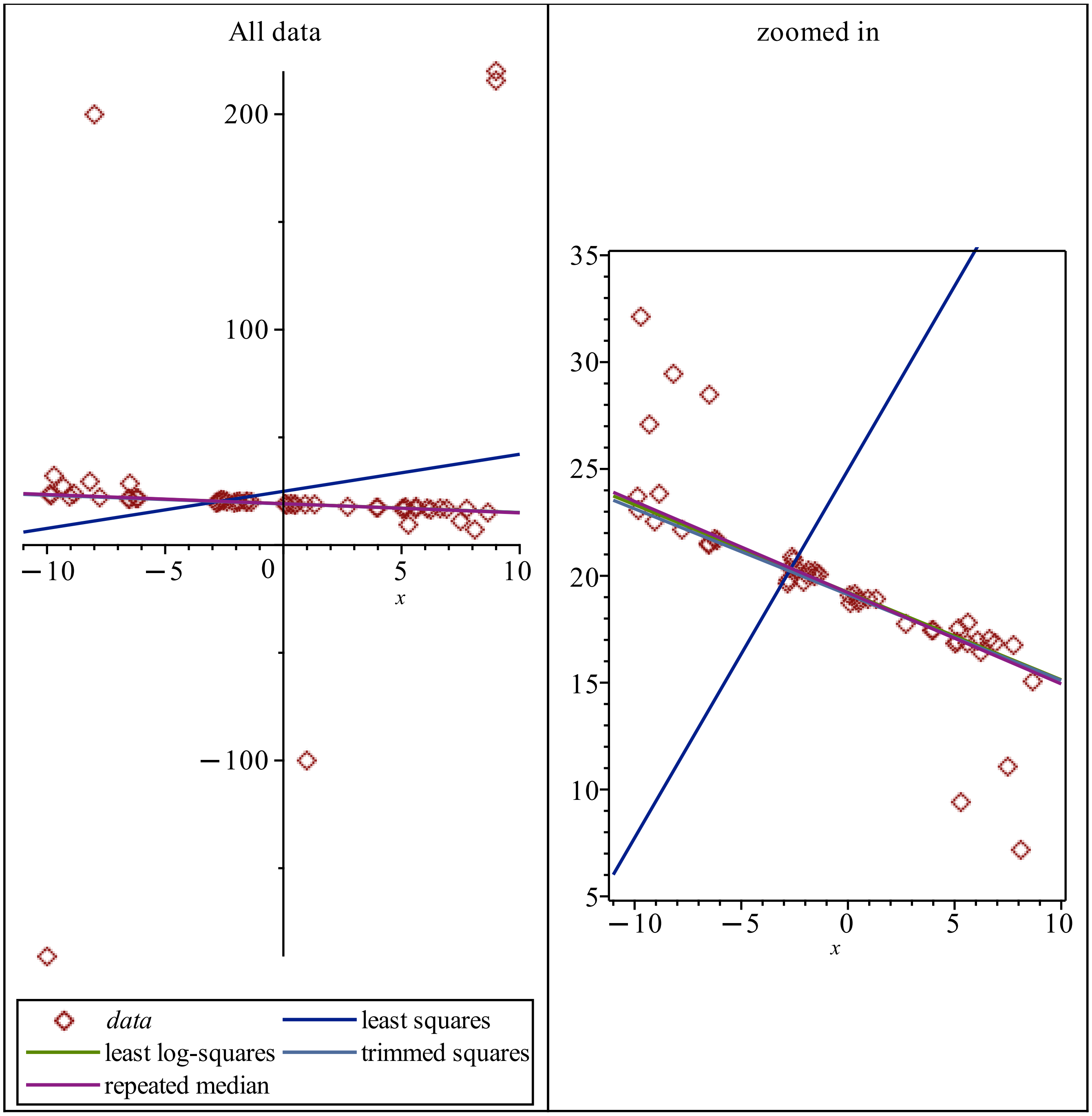
Another robust method is the Repeated Median Estimator (also introduced in the 1980s, but becoming popular recently now that some significant theoretical developments have drastically reduced the computation cost). This looks at the median of the medians of slopes and intercepts over a bunch of pairs of points. I discussed this a little in class, but too much to write here.

```
> rmline := Statistics[RepeatedMedianEstimator](bdata, x)
```

$$rmline := 19.2125102680044 - 0.427092381602868\, x \quad (19)$$

Now let's look at all ov these various versions, and notice that the robust ones are all close, and regular least squares gets strongly influenced by the outliers.

```
> plots[display](<
      plot([bdata, bline, GoodLine, trimline, rmline],
           x = -11 .. 10, style = [point, line $ 4], symbolsize = 19,
  title = "All data",
          legend = [data, "least squares", "least log-squares",
  "trimmed squares", "repeated median"]) |
      plot([bdata, bline, GoodLine, trimline, rmline],
           x = -11 .. 10, style = [point, line $ 4], symbolsize = 19,
  view = [-11 .. 10, 5 .. 35],
           scaling=constrained, axes = boxed, title = "zoomed in")>)
```



As you can see, the robust fits are all quite close (although they differ a little), but the regular least squares is vastly different.

We could discuss measures of how good these are, etc. But really, that should be an AMS class, and I'm tired of least squares. I suspect you are, too.