2019-10-01 The last gasp of least squares.

First, let's grab some data from to fit that includes some nasty outliers.

```
> ExecuteFromWeb:=proc(URL::string, {printfile::truefalse:=false})
   local n,m, status, webfile, headers;
   status,webfile,headers:=HTTP[Get](URL):
   if ( HTTP[Code](status) <> "OK") then
      error(HTTP[Code](status),URL);
   fi;
   # now interpret the maple on the web page
   n:=0:
   while (n < length(webfile)) do
     m:=n;
     parse(webfile,statement,lastread='n', offset=n);
     if (printfile) then printf("%s",webfile[m+1..n]); fi;
   od:
  end:
```

```
> ExecuteFromWeb("http://www.math.stonybrook.edu/~scott/mat331.
      fall19/daily/extras/bdata.txt")
```

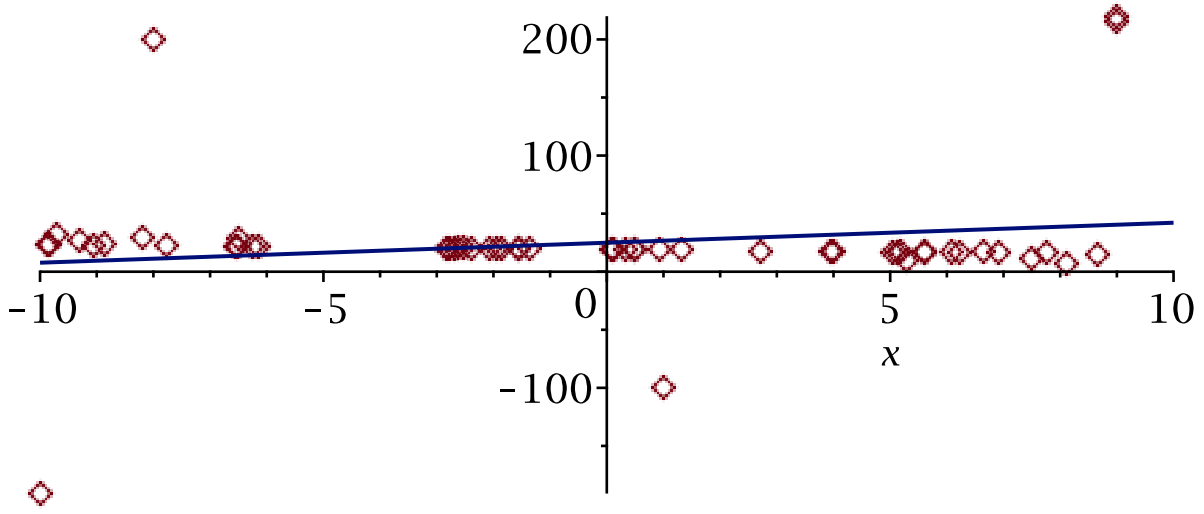
loaded 54 points into bdata.

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

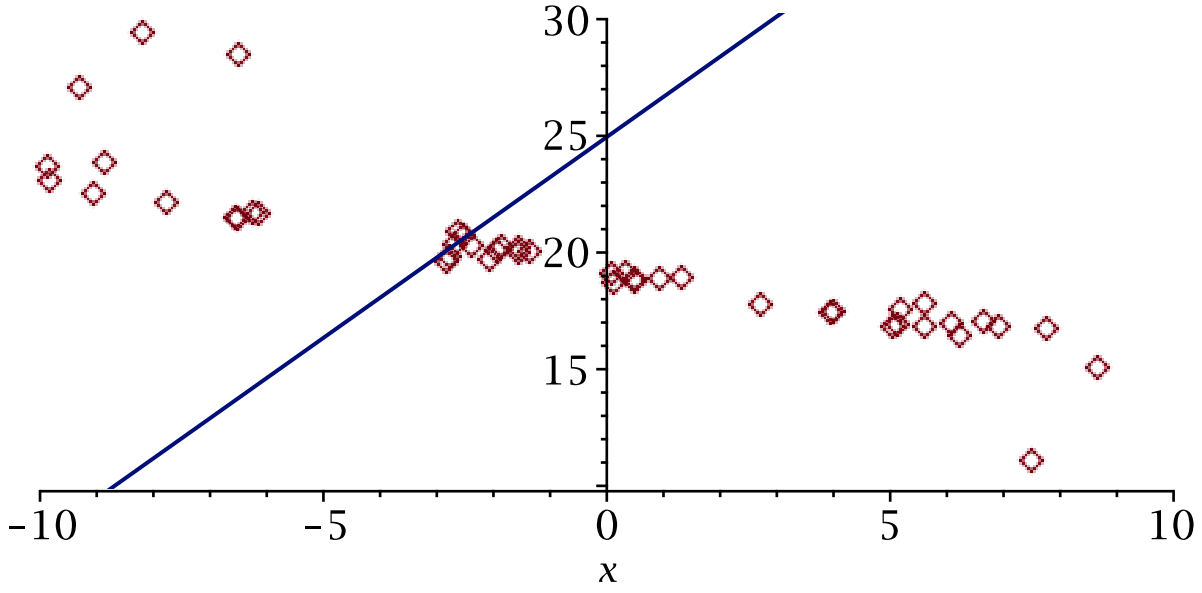
```
> bline := CurveFitting[LeastSquares](bdata, x)
```

$$bline := 24.9502982841362 + 1.72140920411782\,x \quad (1)$$

```
> plot([bdata, bline], x = -10..10, style = [point, line], symbolsize = 19)
```




```
> plot([bdata, bline], x = -10..10, style = [point, line], symbolsize = 19, view = [-10
      ..10, 10..30])
```



Least squares is (apparently) not a great fit, because the outliers screw things up badly.
Let's repeat the process from last time, defining the thing to minimize to be the log of 1+ $(vertical\,dist)^2$. Recall, that is quadratic-like for small distances, but big distances just count like the log... that is, some, but not a crazy amount.

> $obj := (p, m, b) \rightarrow \log(1 + (m \cdot p[1] + b - p[2])^2)$

$$obj := (p, m, b) \mapsto \log\left(1 + \left(m\,p_1 + b - p_2\right)^2\right) \quad (2)$$

> **E:=(data,m,b) -> local i; add(obj(data[i],m,b), i=1..nops(data))**

$$E := (data, m, b) \mapsto add(obj(data_i, m, b), i = 1..nops(data)) \quad (3)$$

Let's guess the slope is -1 and the intercept is 19. How far away is this line?

> $evalf(E(bdata, -1, 19))$

$$139.1518346 \quad (4)$$

Let's compare to the least squares result.

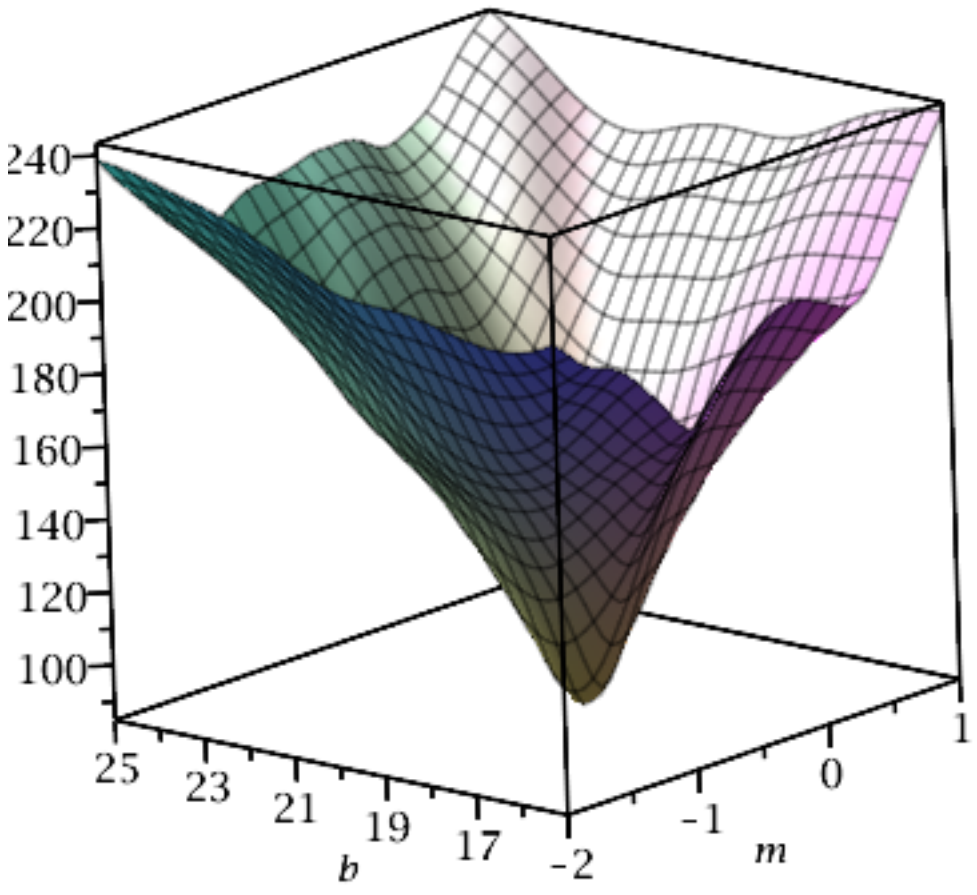
> $evalf(E(bdata, 1.72, 24.95))$

$$254.4789849 \quad (5)$$

So it looks like y=-x+19 is a better fit than what standard least squares gives.

Now let's take a look at what the surface z=E(bdata, m, b) looks like near this line, as we vary slope and intercept.

> $plot3d(E(bdata, m, b), m = -2..1, b = 15..25)$



Clearly there's a unique minimum in the box -2 < m < 1, 15< b < 25.

We could try to find all the critical points of this surface, but it is complicated enough to baffle **solve**, so lets solve numerically.
fsolve needs a little hint (which can either be a range or a starting point. Let's try giving it a starting point:

```
> fsolve( {diff(E(bdata, m, b), m) = 0, diff(E(bdata, m, b), b) = 0}, {m = -1, b
      = 19})
```

$$\{b = -72.09285573,\ m = -26.07452379\} \quad \textbf{(6)}$$

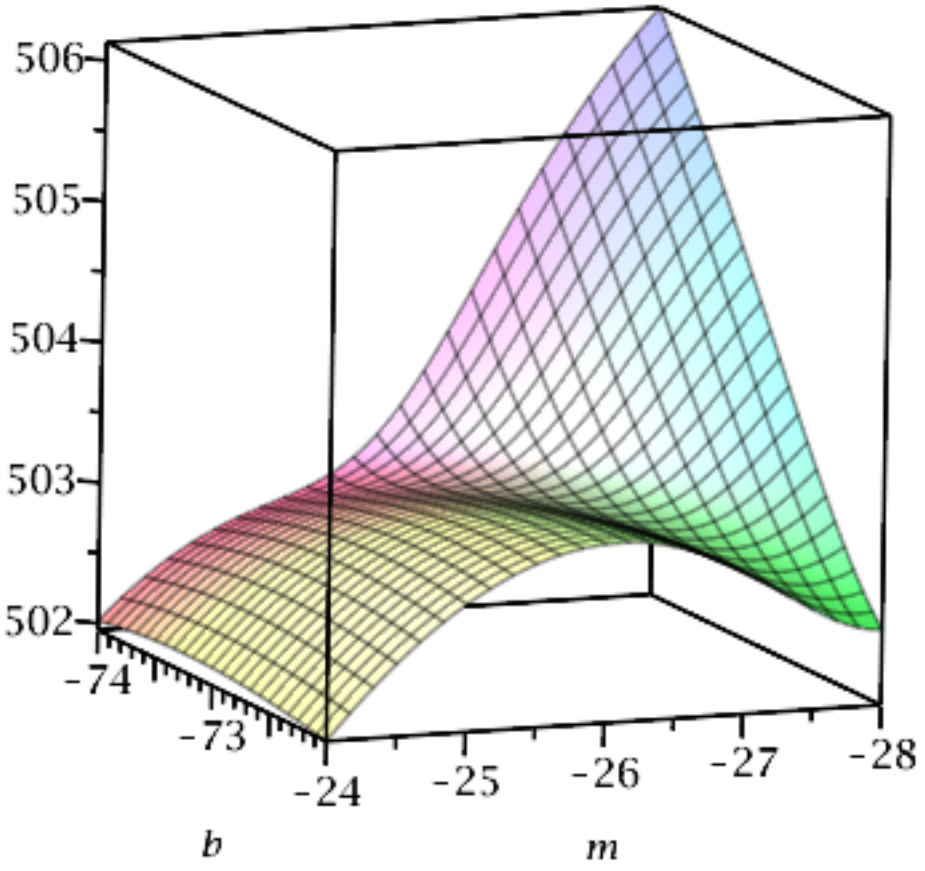
Really? That's not what we expected! Is it a critical point? Sure:

```
> eval( [diff(E(bdata, m, b), m), diff(E(bdata, m, b), b) ], {b = -72.09285573, m
      = -26.07452379});
```

$$[-1.41\ 10^{-8},\ -1.50\ 10^{-8}] \quad \textbf{(7)}$$

But it is a saddle rather than a minimum, as we can see from the graph:

```
> plot3d(E(bdata, m, b), m = -28..-24, b = -74..-72)
```



Since we know roughly where the minimum should be, let's instead try a range for slope and intercept.

> $sol := fsolve(\{diff(E(bdata, m, b), m) = 0, diff(E(bdata, m, b), b) = 0\}, \{m = -1..0, b = 19..20\})$

$$sol := \{b = 19.23067355, m = -0.4096014757\} \quad \textbf{(8)}$$

Yup, that's the line we wanted.

In fact, we didn't discuss this, but Maple has a built-in facility to try to find 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.

> $Optimization[Minimize](E(bdata, m, b))$;

$$[84.2864333974662969, [b = 19.2306735327655, m = -0.4096014803354 49]] \quad \textbf{(9)}$$

This gives us the same solution as above, and also tells us that E≈84.28 there.

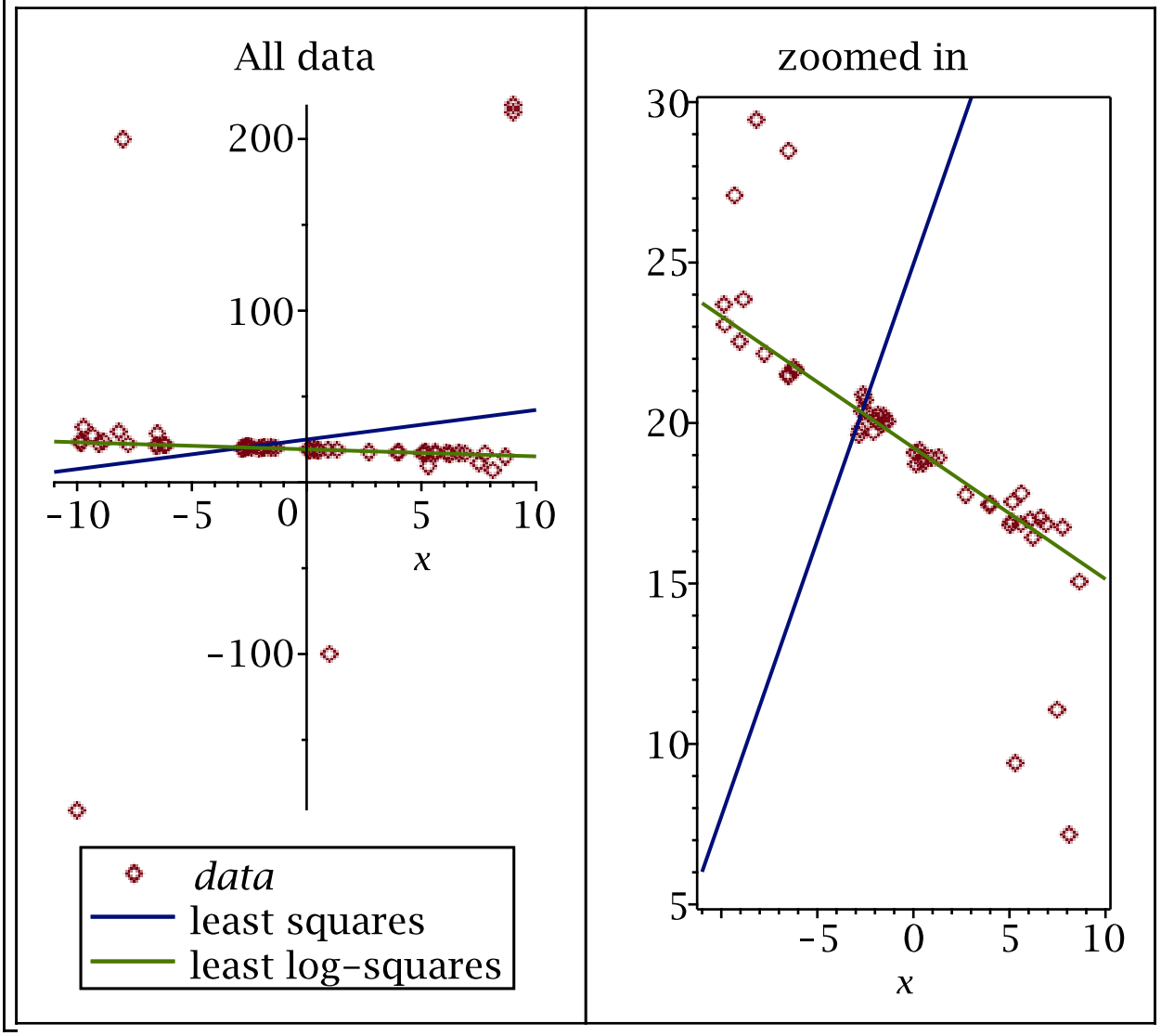
Let's plot the line from minimizing the log(1+x^2) distance and the least squares line along with our points.

```
> lline := subs(sol, m·x + b)
```

$$lline := -0.4096014757\,x + 19.23067355 \quad \textbf{(10)}$$

Let's do both a "wide" view that shows the outliers, and a zoomed in view that shows how different they really are (ie, close to the bulk of the data).

```
> plots[display](⟨
    plot([bdata, bline, lline], x = -11..10, style = [point, line$2], symbolsize = 19,
        title = "All data", legend = [data, "least squares", "least log-squares"])
    |
    plot([bdata, bline, lline], x = -11..10, style = [point, line$2], symbolsize = 19,
        view = [-11..10, 5..30], 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)
    )
```

$$trimline := -0.401442654921944\,x + 19.1216440573069 \quad \textbf{(11)}$$

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

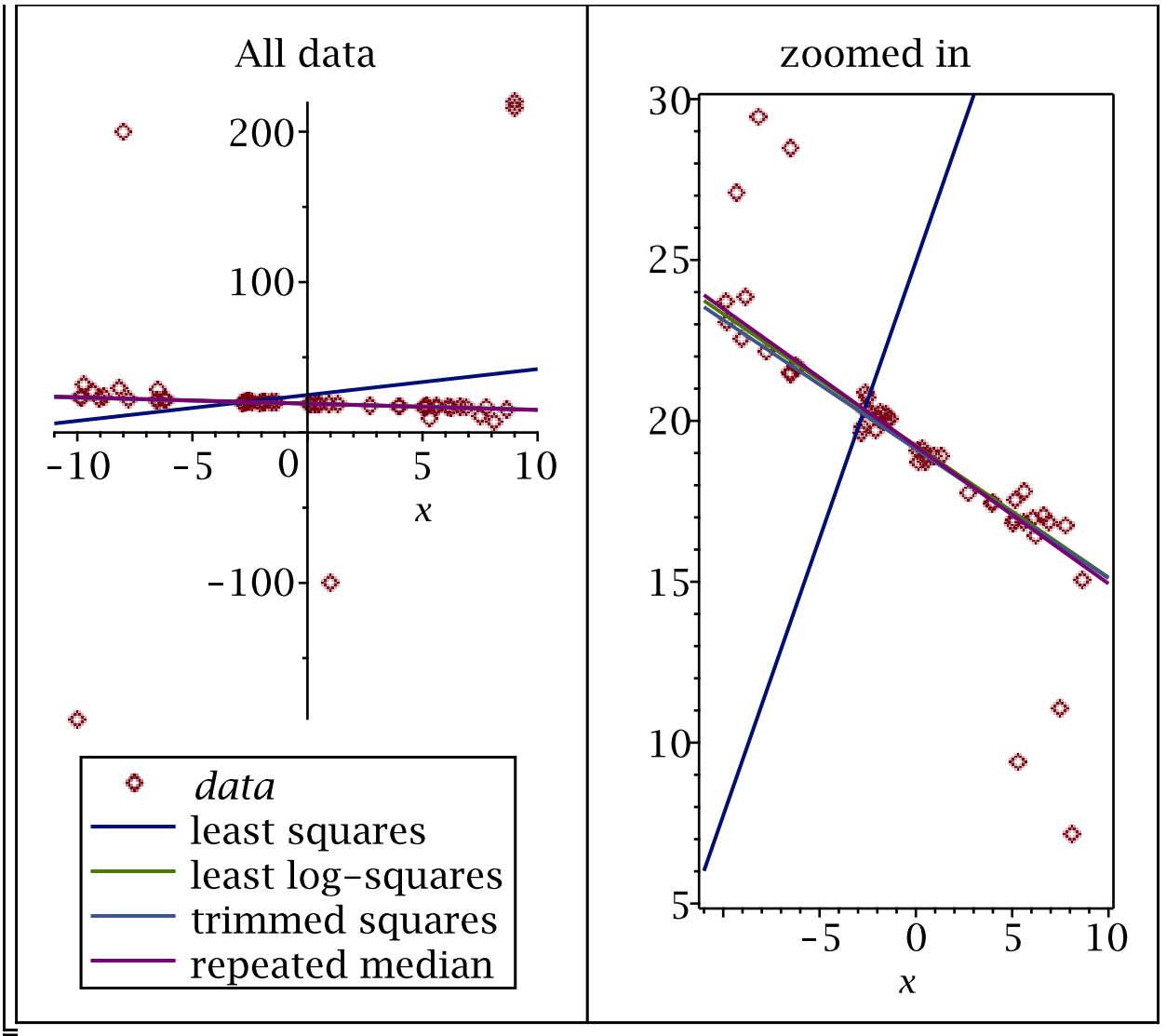
Another robust method is the Repeated Median Estimator (also introduced in the 1980s, but becoming popular recently now that some significant theoretical developments have drastically reduced the computation cost). This looks at the median of the medians of slopes and intercepts over a bunch of pairs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

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

$$rmline := 19.2125102680044 - 0.427092381602868\,x \quad \textbf{(12)}$$

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, lline, 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, lline, trimline, rmline], x = -11..10, style = [point, line$4],
     symbolsize = 19,
         view = [-11..10, 5..30], 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.

Let's move on to talking about differential equations and paper airplanes (gliders). That's not here, so you hadda be there.