I am thinking about the form of the function of sensitivity when the spacing decreases. Does it matter?

If you think your signal is normally distributed, then your standard error is equal to $\sigma = -\frac{2}{\frac{d^2}{dx^2}L(x)}$ the second derivative of your log-likelihood function. (sketch: the Gaussian density is the function that is log-concave downwards, $f(x, \mu, \sigma) = c \exp\left(-\frac{x^2}{\sigma^2}\right)$ so $L(x) = c + -\frac{x^2}{\sigma^2}$ so $\frac{d^2}{dx^2}L(x) = -\frac{2}{\sigma^2}$. (This is how you get standard errors for arbitrary vectors relative to a model fit: compute the Hermetian of your likelihood function, project it onto the vector you want and there you go.)

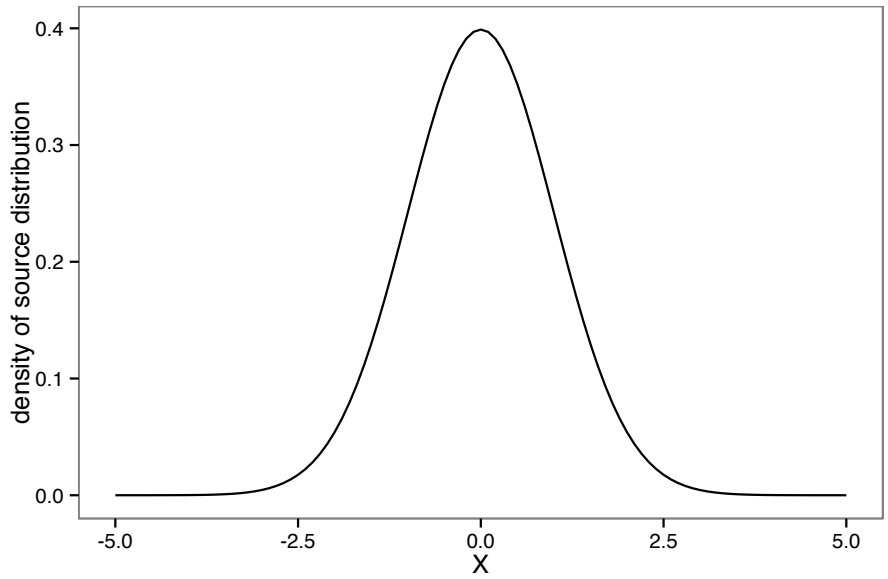
Now, let's run with the idea of the second derivative of the log-likelihood function when there are several objects. You'd have several sigmas, added together; $\Pr(x) = c\sum_{i=-N}^{N} \exp\left(-\frac{(x-is)^2}{\sigma^2}\right)$; so that $L(x)$ would be the log of that.

That's the log of a sum, which resists simplification. What are other ways to cast this problem? Fourier domain?

Okay, let's think about this without trying to do the math. You have a position signal. It comes in with a roughly bell-shaped distribution. You need to correlate it with a later signal. Those distractor signals come in with relatively Gaussian distribution. Assume those distractor signals provide no information about the motion of the real target. So you have a bunch of overlapping signals, some idea about where the signal *was*, and you need to derive where the signal *is* now.

So, let's say your position signal is distributed according to a bell-shaped function.

```
(fplot(dnorm, c(-5,5)) + xlab("X")
 + ylab("density of source distribution"))
```




```
## Note: no visible binding for '<<-' assignment to 'scales'
```

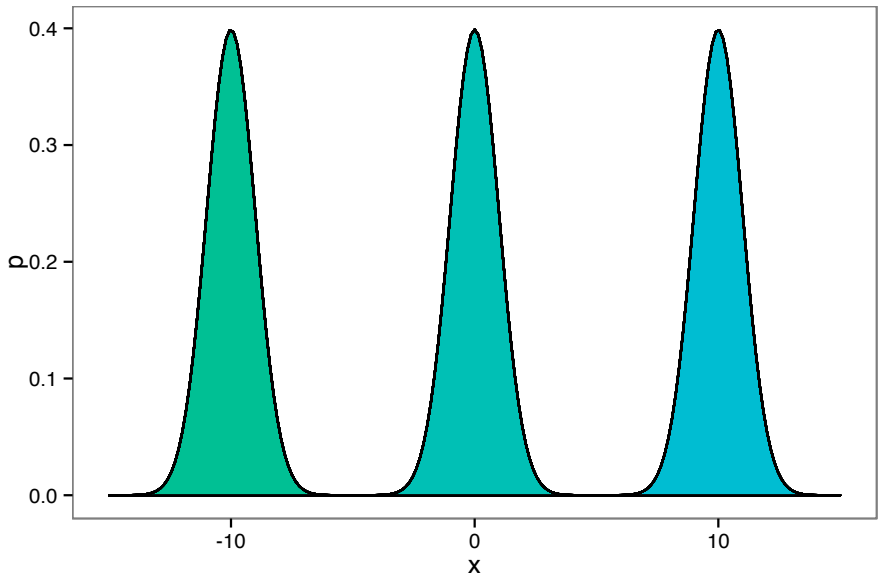
Let's consider all the signals coming from all the distractor objects; they're distributed identically. Here I'll make a dataset with all of their signal densities.

```
dataset <- function(spacing=1, sigma=1, sources=-10:10,
                    sampling=seq(-15, 15, length=201)) {
  r <- expand.grid(x=sampling, sourceid=sources)
  mutate(r, sourceloc=sourceid*spacing, p=dnorm(x, sourceloc, sigma))
}
```

If we plot the densities stacked on top of each other, it looks like this.

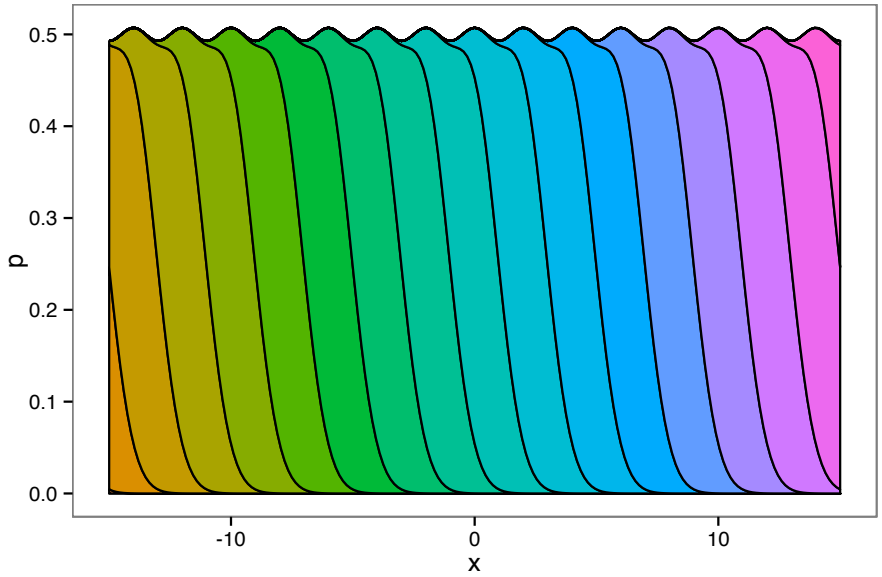
```
(ggplot(dataset(spacing=10), aes(x, p, fill=factor(sourceid)))
 + geom_area(color="black")
 + opts(legend.position="none")
 )

## 'opts' is deprecated.  Use 'theme' instead.  (Deprecated; last used
in version 0.9.1)
```



And bringing them closer together:

```
(ggplot(dataset(spacing=2), aes(x, p, fill=factor(sourceid)))
 + geom_area(color="black")
 + theme(legend.position="none")
 )
```



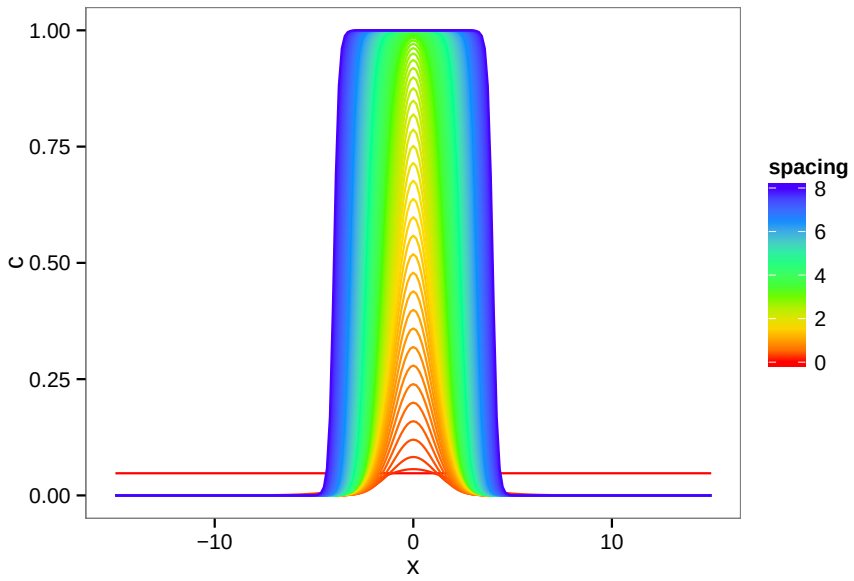
At some point the correct signal gets lost among the other distractor signals.

Now let's try computing the "probability of correct identification." That is, supposing you have some idea where you're looking (x), and a bunch of other signals, what is the probability that you come up with the correct signal (here, the one with id=0)?

```
correct.identification <- function(data=dataset(...), ...) {
  d <- ddply(data, "x", summarize, c=p[which(sourceid==0)]/sum(p), total=sum(p))
  mutate(d, total=total/max(total))
}
```

Computing this "correct identification" over a range of spacings. Here each colored line corresponds to a different spacing.

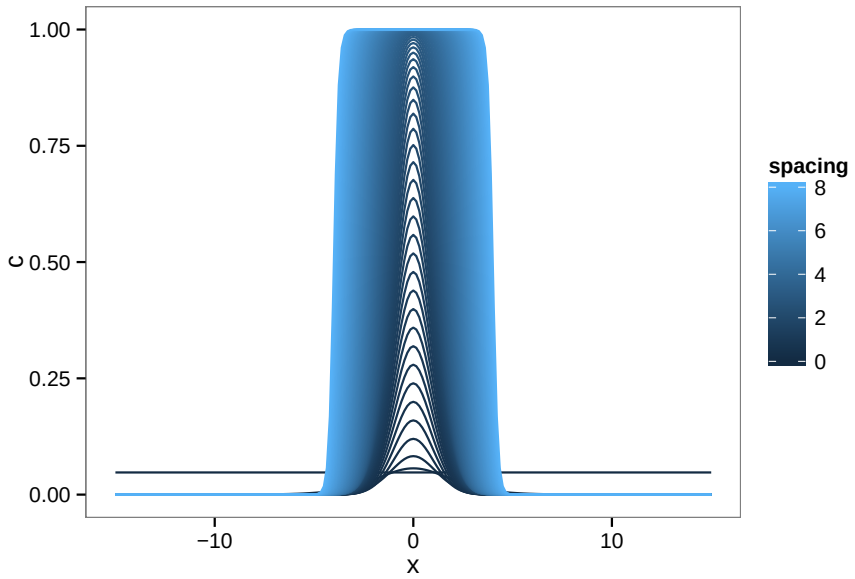
```
cid <- mdply(data.frame(spacing=seq(0,8,0.1)), correct.identification)
(ggplot(cid, aes(x, c, color=spacing, group=spacing))
  + geom_line() + scale_color_gradientn(colours=rainbow(7)[-7]))
```



So, as distractors come closer, the range over which you can correctly identify the corresponding signal shrinks and then the probability of identification decreases.

Interchanging X, we see the probability of correct identification as a function of spacing, with the colored lines corresponding to the amount of error in the initial position signal.

```
ggplot(cid, aes(x, c, color=spacing, group=spacing)) + geom_line()
```



After this, I have a notion about iterating this process; what happens when you have a small probability of misbinding, but it's repeated over and over again? This may result in the "collapse" when spacing is below the critical distance, where sensitivity drops to near zero (rather than smoothly declining). This may be lead to viewing the motion detecting process as a sort of "particle filter" algorithm, where each particle updates subject to a certain distance of summation.