Stimuli and Simulated Interpretations

Alex Kale

2020-11-19

Synthetic data

We start by creating a grid of experimental manipulations and controls representing the space of possible comparisons we might use to differentiate between alternative processes for graphical statistical inference. In our experiments, we will run a subset of these manipulations that maximize our ability to differentiate between alternative inference processes.

```
# set up grid of data conditions
first_year <- 1990
last_year <- 2015
n_years <- last_year - first_year + 1
mid_year <- first_year + n_years / 2
n_draws <- 50

df <- expand_grid(
  "year" = seq(first_year, last_year, length.out = n_years),
  "intercept" = 100,
  "ref_slope" = 1,
  "target_slope_diff" = seq(0, 1, length.out = 5),
  "sd" = seq(1, 10, length.out = 4)#,
  # "axis_limits" = list(c(60, 140), c(20, 180)),
  # "aspect_ratio" = c(1, 2),
  # "ensemble_size" = c(2, 3, 5, 7, 10, 20, n_draws)
)

head(df)
```

```
## # A tibble: 6 x 5
##    year intercept ref_slope target_slope_diff    sd
##   <dbl>     <dbl>     <dbl>              <dbl> <dbl>
## 1  1990       100         1               0        1
## 2  1990       100         1               0        4
## 3  1990       100         1               0        7
## 4  1990       100         1               0       10
## 5  1990       100         1               0.25     1
## 6  1990       100         1               0.25     4
```

Using the grid of manipulations above, we generate the data we will visualize as stimuli.

```
# generate target line, 95% CI of reference distribution, and draws from reference
df = df %>%
  mutate(
    # target line
    target = intercept + (ref_slope + target_slope_diff) * (year - mid_year) + sd * map_
dbl(year, ~rnorm(1)),
    # confidence interval for reference distribution
    ref_avg = intercept + ref_slope * (year - mid_year),
    ref_lb = intercept + ref_slope * (year - mid_year) + qnorm(0.025, mean = 0, sd = s
d),
    ref_ub = intercept + ref_slope * (year - mid_year) + qnorm(0.975, mean = 0, sd = s
d),
    # draws from reference distribution
    draw = list(seq(1, n_draws, length.out = n_draws)),
  ) %>%
  unnest(cols = c("draw")) %>%
  mutate(
    # compute reference lines
    noise = sd * map_dbl(draw, ~rnorm(1)),
    ref = intercept + (ref_slope) * (year - mid_year) + noise
  )

head(df)
```

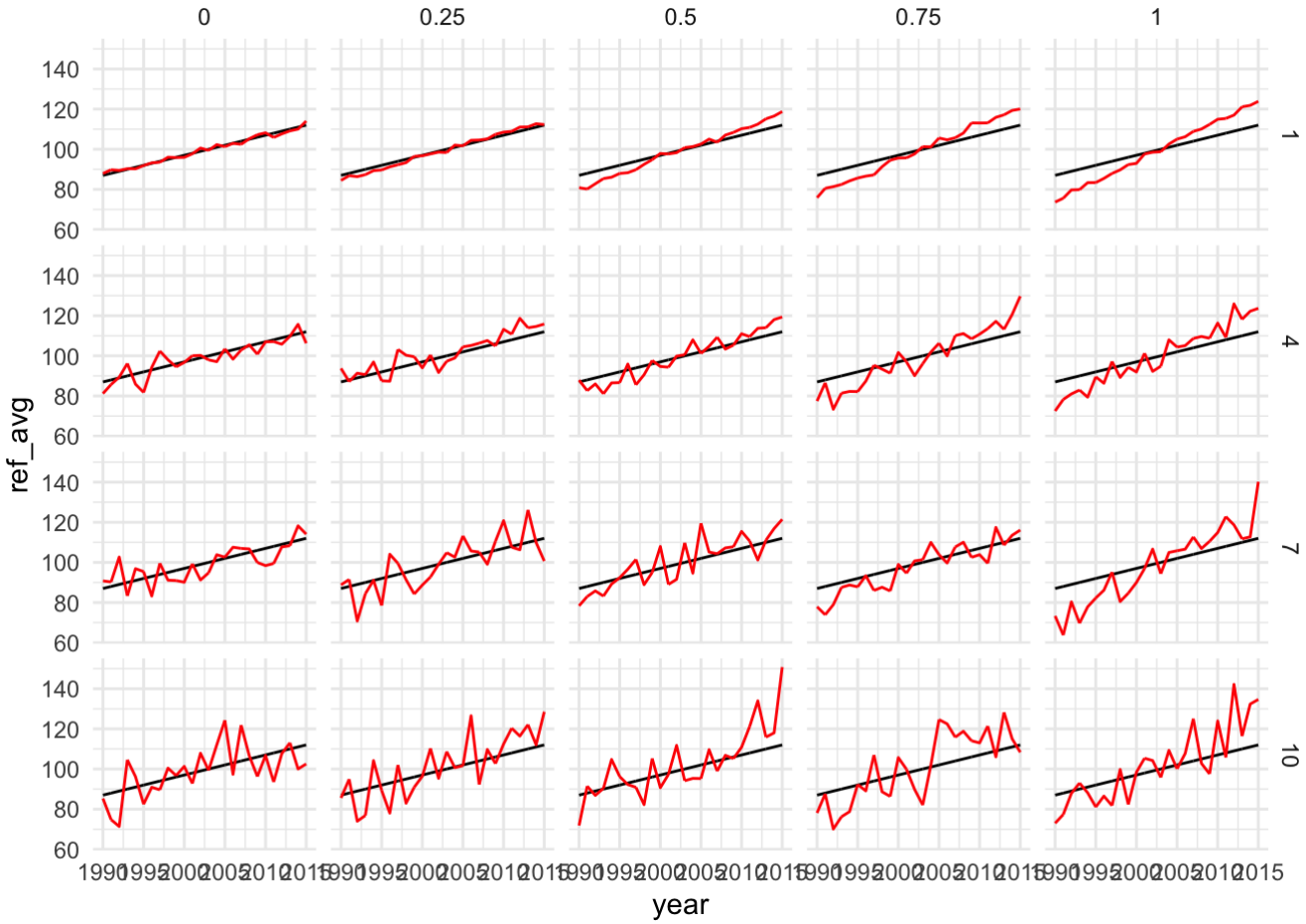
```
## # A tibble: 6 x 12
##    year intercept ref_slope target_slope_di…    sd target ref_avg ref_lb ref_ub
##   <dbl>     <dbl>     <dbl>            <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl>
## 1  1990       100         1                0     1   87.9      87   85.0   89.0
## 2  1990       100         1                0     1   87.9      87   85.0   89.0
## 3  1990       100         1                0     1   87.9      87   85.0   89.0
## 4  1990       100         1                0     1   87.9      87   85.0   89.0
## 5  1990       100         1                0     1   87.9      87   85.0   89.0
## 6  1990       100         1                0     1   87.9      87   85.0   89.0
## # … with 3 more variables: draw <dbl>, noise <dbl>, ref <dbl>
```

Possible stimuli

Each of the following grid of possible stimuli represent a different point in the design space for conveying reference uncertainty. Here, I omit stimulus variations with difference axis scales, aspect ratios, and ensemble sizes for brevity, although later we will show how varying these attributes of chart design map help us differentiate between possible inference processes.

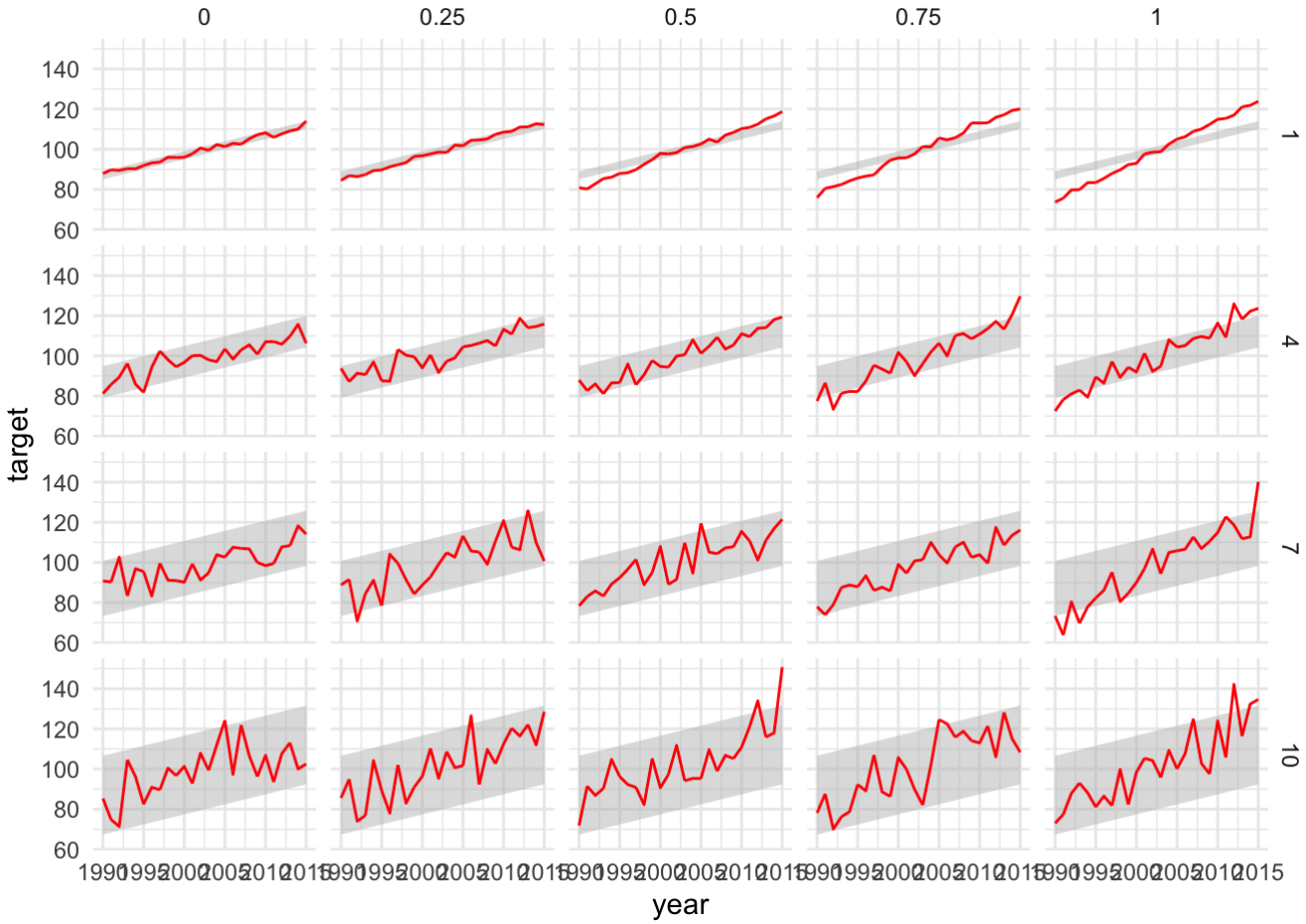
First, we have the design strategy of showing no uncertainty in the reference distribution.

```
# no reference uncertainty
df %>%
  ggplot(aes(x = year)) +
  geom_line(aes(y = ref_avg)) +
  geom_line(aes(y = target), color = "red") +
  theme_minimal() +
  facet_grid(sd ~ target_slope_diff)
```



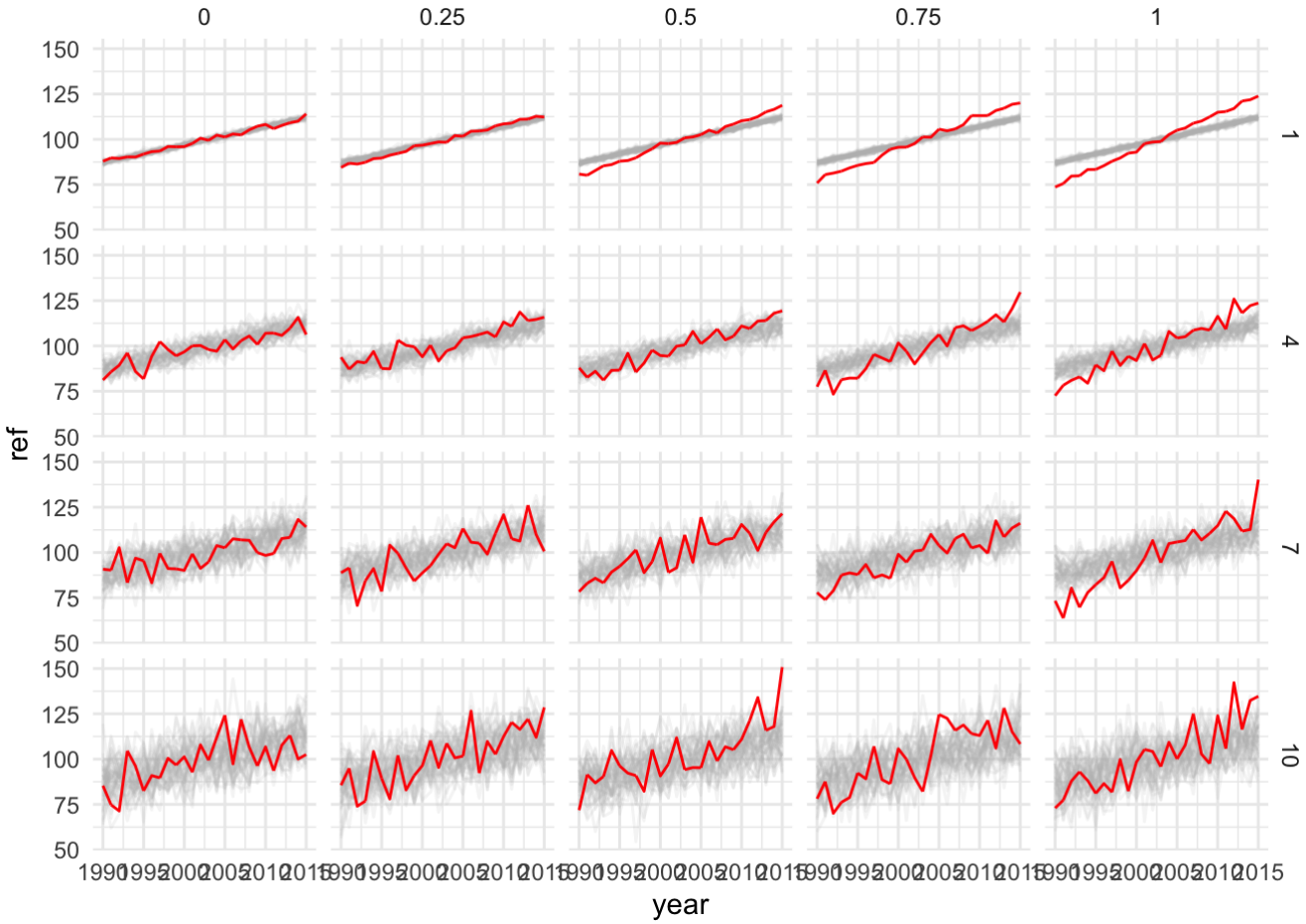
Second, we have the design strategy of showing uncertainty in the reference distribution as an envelope containing 95% of possible lines under the reference model.

```
# envelope of reference uncertainty
df %>%
  ggplot(aes(x = year)) +
  geom_ribbon(aes(ymin = ref_lb, ymax = ref_ub), fill = "gray", alpha = 0.5) +
  geom_line(aes(y = target), color = "red") +
  theme_minimal() +
  facet_grid(sd ~ target_slope_diff)
```



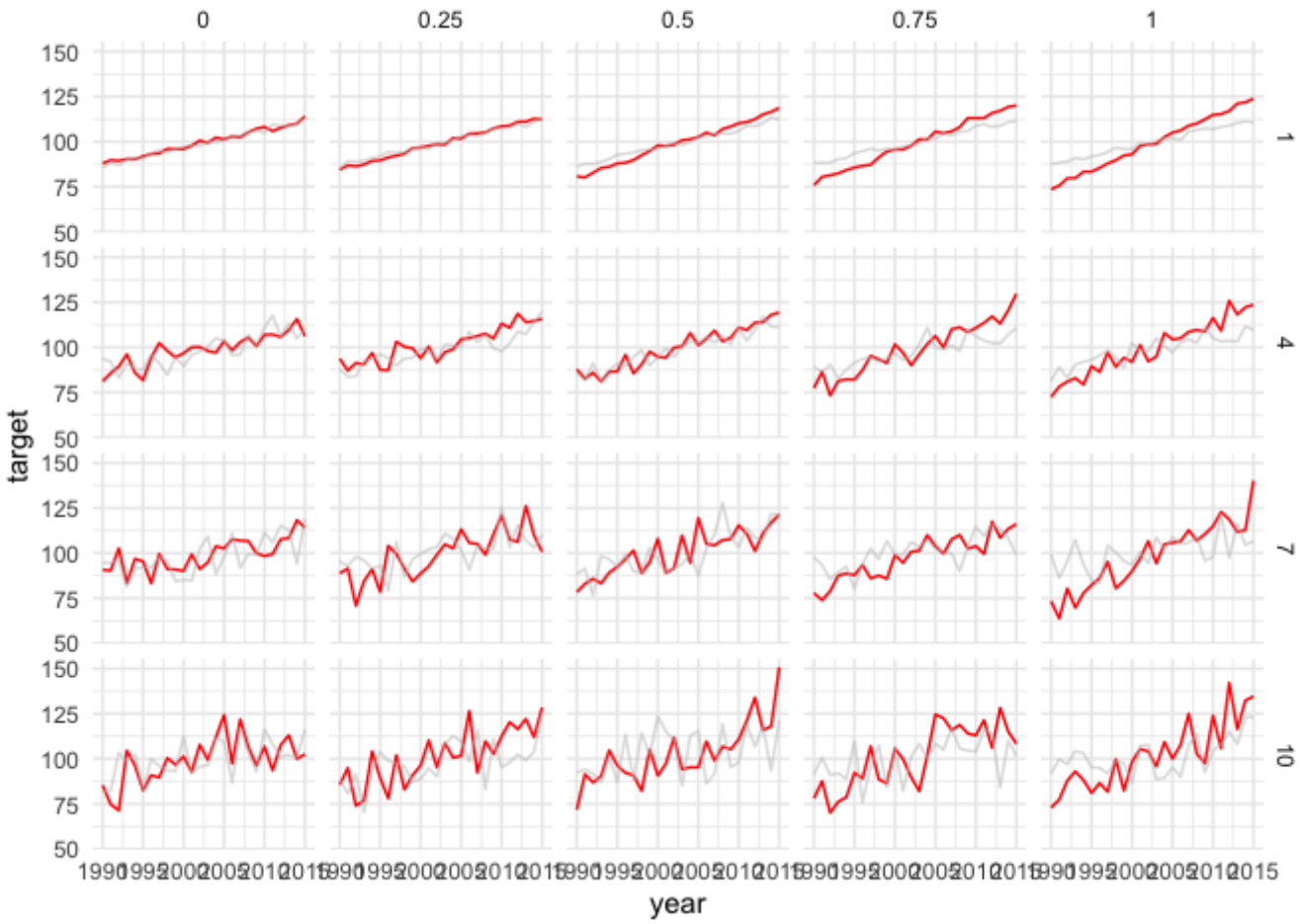
Third, we have the strategy of showing uncertainty in reference distribution as a static ensemble of draws.

```
# static ensemble of reference uncertainty
df %>%
  ggplot(aes(x = year)) +
  geom_line(aes(y = ref, group = draw), alpha = .15, color = "gray", show.legend = FALSE
) +
  geom_line(aes(y = target), color = "red") +
  theme_minimal() +
  facet_grid(sd ~ target_slope_diff)
```



Last, we show reference uncertainty as HOPs.

```
# HOPs of reference uncertainty
plt_spec <- df %>%
  ggplot(aes(x = year)) +
  geom_line(aes(y = target), color = "red") +
  geom_line(aes(y = ref), alpha = .5, color = "gray", show.legend = FALSE) +
  theme_minimal() +
  facet_grid(sd ~ target_slope_diff) +
  transition_manual(draw)

animate(plt_spec, fps = 2.5, duration = 20)
```



Alternative inference processes

We define alternative inference processes as model checks (following Gelman). The basic idea is that inference involves some sort of visual comparison between a target and a reference distribution. This comparison can take on different forms. In this study, we break down model checks based on possible *visual test statistics* `T(.)` and possible *discrepancy functions* `g()` such that the former represents the visual feature or heuristic that is used to inform the chart user's sense of signal in a chart, and the later represents the cognitive process that acts on that signal. By making a distinction between these two aspects of graphical statistical inference, we are able to focus our investigation primarily on cognitive processes and sidestep well-known difficulties in determining which of a set of highly correlated visual cues (a.k.a., perceptual proxies) a chart user is relying on.

Visual test statistics (a.k.a., perceptual proxies)

We assume that slope is the fundamental visual test statistic for comparing trends. In actuality, the test statistic could be average value on the y-axis, range on the y-axis, or any number of correlated proxies. The perceptual proxy for comparison is unimportant to us. The way these proxies are used to construct *different forms of visual statistical test* is our primary concern.

```
# slope function (visual test statistic)
# (expects vectors of cartesian coordinates)
slope <- function(x, y) {
  # lagged differences
  dx <- diff(x)
  dy <- diff(y)

  return(mean(dy / dx))
}
```

Discrepancy functions

The primary purpose of this study is to investigate when and how often chart users rely on different forms of visual statistical test to make inferential interpretations of the signal in a chart. We call these alternative forms of graphical statistical inference *discrepancy functions*. Here we define some candidate discrepancy functions that we will compare and use below to simulate interpretations of the signal in a chart.

We want to simulate the sense of signal on a scale from 0 to 1, mapping roughly to perceived probability that the target was generated by a different underlying process than the reference distribution. To match the units of signal across discrepancy functions we'll need to normalize the output of each discrepancy function. However, different forms of discrepancy functions entail different ways of interpreting the minimum and maximum signal that could be conveyed by a chart.

```
# normalization to match scale of discrepancy functions
normalize <- function(signal, minimum, maximum) {
  return((signal - minimum) / (maximum - minimum))
}
```

The simplest form of discrepancy function is a difference between the visual test statistic for the target and average of the reference distribution. There are multiple alternative ways of normalizing the difference between visual test statistics, and these alternative normalizations entail different discrepancy functions based on the difference between visual test statistics.

```
# difference between visual test statistics
# (expects single reference vector, e.g., avg of ensemble)
difference <- function(x, target, reference, T) {
  T_target <- T(x, target)
  T_reference <- T(x, reference)

  return(T_target - T_reference)
}
```

Another discrepancy function of interest measures the overlap between target and reference, in this case the proportion of the target line that is not contained by the reference distribution. Notice that unlike the difference function, this alternative process does not apply a visual test statistic `T(.)` to both target and reference separately but instead compares them directly. Although this discrepancy function deviates slightly from Gelman's formalism, it seems like a distinct possibility worth considering. In the univariate case (rather than line charts), the analogous discrepancy function would be the proportion of overlap between to distributions.

```
# proportion not contained
# (expects single reference vectors, e.g., min and max of ensemble)
proportion_not_contained <- function(target, reference_lb, reference_ub) {
  # discrepancy is how much of target falls outside of reference range
  return(sum(target < reference_lb || target > reference_ub) / length(target))
}
```

Some discrepancy functions are approximations of *likelihood* estimation. These fall under the umbrella of *Bayesian cognition* in that they involve sampling processes through which a chart user's prior could have an influence of the sense of signal.

The first likelihood approximation we consider is based on the percentile of target within the empirical cumulative density function of the reference distribution.

```
# approximation of likelihood based on percentile within empirical cumulative density fu
nction
# (expects list of reference draws at each x value, e.g., an ensemble)
empirical_cdf <- function(x, target, reference, n, T) {
  # calculate slopes for target and reference distribution
  T_target <- T(x, target)
  T_ref <- c()
  for(i in 1:n) { # iterate over draws calculating slopes for each
    draw_ref = unlist(map(reference, ~.[[i]]))
    T_ref = append(T_ref, T(x, draw_ref))
  }
  # percentile of target within reference
  pr <- ecdf(T_ref)
  percentile <- pr(T_target)

  # transform so that smallest value represents smallest signal for different data-gener
ating processes
  return(abs(percentile - 0.5))
}
```

The second likelihood-based discrepancy function consider is essentially a t-test. We take the absolute value of the t-statistic as a measure of how likely the target is under the reference process.

```
# likelihood-based t-test
# (expects list of reference draws at each x value, e.g., an ensemble)
t_test <- function(x, target, reference, n, T) {
  # calculate slopes for target and reference distribution
  T_target <- T(x, target)
  T_ref <- c()
  for(i in 1:n) { # iterate over draws calculating slopes for each
    draw_ref = unlist(map(reference, ~.[[i]]))
    T_ref = append(T_ref, T(x, draw_ref))
  }
  # calculate mean and sampling error for slopes based on reference distribution
  T_ref_avg <- mean(T_ref)
  T_ref_var <- var(T_ref)
  se <- sqrt(T_ref_var / n)
  # calculate t statistics (expresses likelihood)
  t <- (T_target - T_ref_avg) / se

  return(abs(t))
}
```

Simulation

We pass each data condition through the discrepancy functions defined above to generate possible user interpretations.

First, we set up consistent color scale across each alternative cognitive process.

Next, we'll want to convert from data units into pixels so that predicted responses can account for factors like axis scale and aspect ratio.

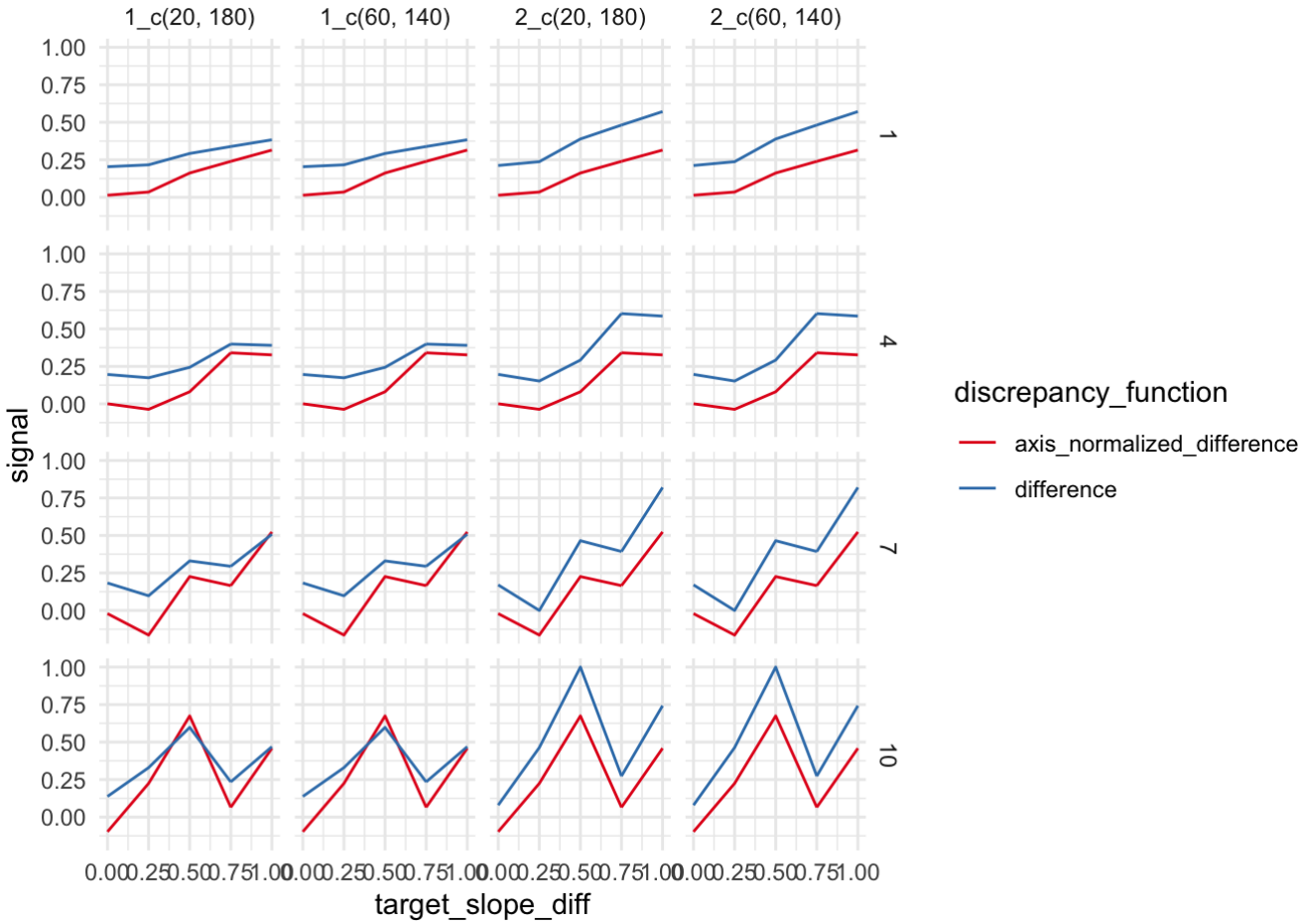
```
chart_width_pixels <- 500

simulation_df <- df %>%
  # add axis scale and aspect ratio as design manipulations in our dataframe
  expand_grid(aspect_ratio = c(1, 2), axis_limits = list(c(60, 140), c(20, 180))) %>%
  # convert all discrepancy function inputs into pixel units (roughly D3-style interpola
tion)
  mutate(
    year = (year - first_year) / (last_year - first_year) * chart_width_pixels,
    target = (target - unlist(axis_limits)[1]) / (unlist(axis_limits)[2] - unlist(axis_l
imits)[1]) * chart_width_pixels * aspect_ratio,
    ref_avg = (ref_avg - unlist(axis_limits)[1]) / (unlist(axis_limits)[2] - unlist(axis
_limits)[1]) * chart_width_pixels * aspect_ratio,
    ref_lb = (ref_lb - unlist(axis_limits)[1]) / (unlist(axis_limits)[2] - unlist(axis_l
imits)[1]) * chart_width_pixels * aspect_ratio,
    ref_ub = (ref_ub - unlist(axis_limits)[1]) / (unlist(axis_limits)[2] - unlist(axis_l
imits)[1]) * chart_width_pixels * aspect_ratio,
    ref = (ref - unlist(axis_limits)[1]) / (unlist(axis_limits)[2] - unlist(axis_limits)
[1]) * chart_width_pixels * aspect_ratio
  )
```

Now, we generate possible interpretations separately for each type of uncertainty visualization.

When we *don't visualize reference uncertainty*, all possible discrepancy functions are variations of the difference function with normalization either implied by the range of observed signals or the design of the axis scale (corresponding to Kosslyn's "inner vs outer framework").

```
simulation_df %>%
  group_by(target_slope_diff, sd, aspect_ratio, axis_limits) %>%
  summarise(
    # inputs
    year = unique(year),
    target = unique(target),
    ref_avg = unique(ref_avg),
    # discrepancy functions
    g_difference = difference(year, target, ref_avg, slope),
  ) %>%
  ungroup() %>%
  mutate(
    # normalization implied by axis scale (range from no slope to aspect ratio of chart)
    g_axis_normalized_difference = normalize(g_difference, 0, aspect_ratio),
    # normalization implied by observed signals
    g_difference = normalize(g_difference, min(g_difference), max(g_difference))
  ) %>%
  pivot_longer(
    cols = starts_with("g_"),
    names_to = "discrepancy_function",
    names_prefix = "g_",
    values_to = "signal",
    values_drop_na = TRUE
  ) %>%
  unite("scale", aspect_ratio:axis_limits, remove = FALSE) %>%
  ggplot(aes(x = target_slope_diff, y = signal, color = discrepancy_function)) +
  geom_line() +
  colScale +
  theme_minimal() +
  facet_grid(sd ~ scale)
```

```
## `summarise()` regrouping output by 'target_slope_diff', 'sd', 'aspect_ratio', 'axis_l
imits' (override with `.groups` argument)
```

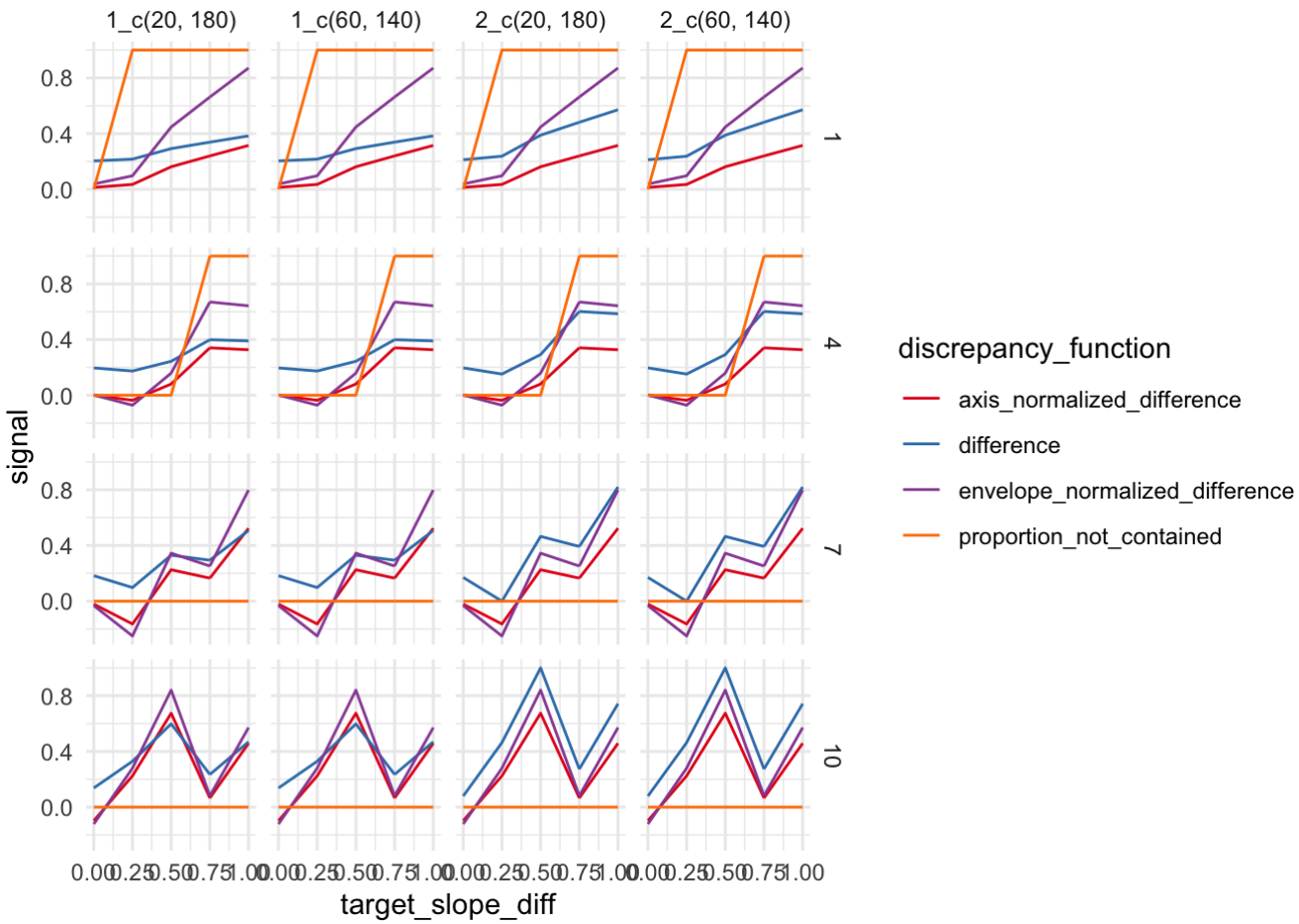


These charts show simulations of perceived signal as a function of the ground truth difference in slope between the target and reference. Rows show different levels of variance in the data-generating process. Columns show different combinations of aspect ratio (square or tall) of y-axis limits (loose or tight). The idea behind manipulating aspect ratio and y-axis limits is that the distances in pixels on the chart should be the same for a square chart with tight axis limits `1_c(60, 140)` and a tall chart with loose axis limits `2_c(20, 180)`, yet discrepancy functions normalized differently will make different predictions for these cases. We will create grids of charts like this to show which manipulations are necessary to distinguish between specific candidate discrepancy functions.

When we *visualize an envelope of reference uncertainty*, possible discrepancy functions are three variations of the difference function—with normalization implied by (1) the range of observed signals, (2) the design of the axis scale, or (3) the range of signals suggested by the envelope of uncertainty in a given chart—and a new discrepancy function based on the proportion of the target contained within the reference envelope. Not that the discrepancy function based on proportion contained does not rely on a visual test statistic `T(.)` and represents a plausible departure from Gelman's formulation of discrepancy functions as `T(target) - T(reference)`.

```
simulation_df %>%
  group_by(target_slope_diff, sd, aspect_ratio, axis_limits) %>%
  summarise(
    # inputs
    year = unique(year),
    target = unique(target),
    ref_avg = unique(ref_avg),
    ref_lb = unique(ref_lb),
    ref_ub = unique(ref_ub),
    # discrepancy functions
    g_difference = difference(year, target, ref_avg, slope),
    g_proportion_not_contained = proportion_not_contained(target, ref_lb, ref_ub),
    # auxiliary inputs to normalization
    min_envelope = min(ref_lb),
    max_envelope = max(ref_ub)
  ) %>%
  ungroup() %>%
  mutate(
    # normalization implied by axis scale (range from no slope to aspect ratio of chart)
    g_axis_normalized_difference = normalize(g_difference, 0, aspect_ratio),
    # normalization implied by signals in envelope (range from no slope to max slope in
 envelope)
    g_envelope_normalized_difference = normalize(g_difference, 0, (max_envelope - min_en
velope) / chart_width_pixels),
    # normalization implied by observed signals
    g_difference = normalize(g_difference, min(g_difference), max(g_difference)),
    g_proportion_not_contained = normalize(g_proportion_not_contained, min(g_proportion_
not_contained), max(g_proportion_not_contained))
  ) %>%
  pivot_longer(
    cols = starts_with("g_"),
    names_to = "discrepancy_function",
    names_prefix = "g_",
    values_to = "signal",
    values_drop_na = TRUE
  ) %>%
  unite("scale", aspect_ratio:axis_limits, remove = FALSE) %>%
  ggplot(aes(x = target_slope_diff, y = signal, color = discrepancy_function)) +
  geom_line() +
  colScale +
  theme_minimal() +
  facet_grid(sd ~ scale)
```

```
## `summarise()` regrouping output by 'target_slope_diff', 'sd', 'aspect_ratio', 'axis_l
imits' (override with `.groups` argument)
```



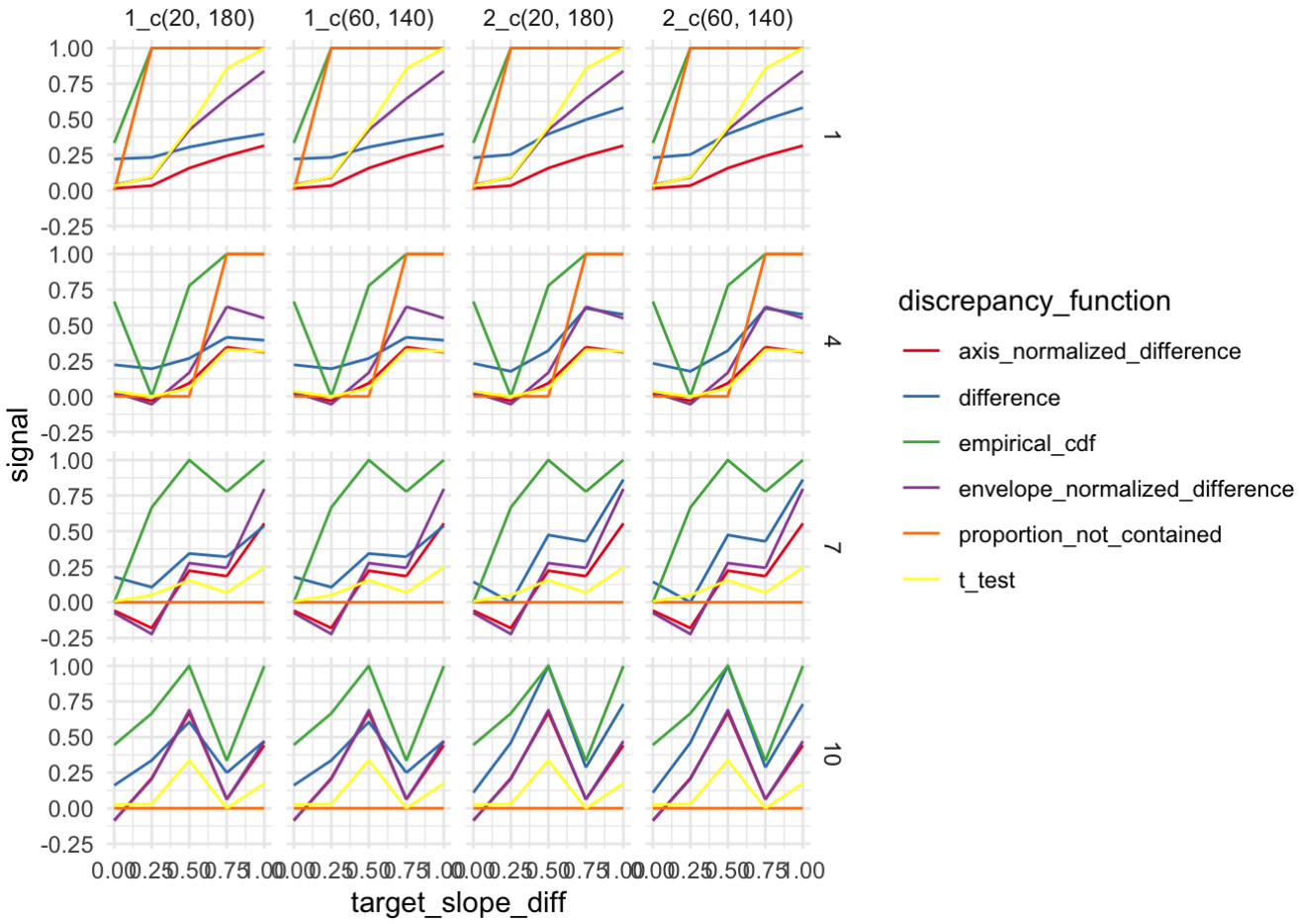
We can see that adding an envelope of uncertainty results in more possible interpretations of the signal in the chart. The two new discrepancy functions we've added are based on the envelope which is itself dependent on the variance on the data generating process. We can see that when variance is relatively high, the sense of signal collapses to zero for all discrepancy functions but difference. We can also see that a proportion not contained discrepancy function lends itself to dichotomous interpretations more than the other strategies.

When we *visualize an ensemble of reference uncertainty* (static ensemble or HOPs), possible discrepancy functions include the four defined above—but now where ensemble average and envelope are approximated from an ensemble—as well as two additional discrepancy functions that only become possible when the reference distribution is shown as an ensemble. These are an empirical cumulative density function (cdf) and a visual t-test, both based on a sampled subset of the ensemble.

```
sample <- 20 # we can change sample size for empircal cdf and t-test (1:50)

simulation_df %>%
  # for ensembles, we define inputs to discrepancy functions based on ensemble processin
g rather than direct value decoding
  group_by(target_slope_diff, sd, aspect_ratio, axis_limits, year) %>%
  summarise(
    # inputs
    target = unique(target),
    ref_avg = mean(ref), # calculate ensemble average and boundaries to approximate inpu
ts used for previous charts
    ref_lb = min(ref),
    ref_ub = max(ref),
    draw = list(draw), # process ensembles as lists of possible values at each year
    ref = list(ref)
  ) %>%
  mutate(
    # discrepancy functions
    g_difference = difference(year, target, ref_avg, slope),
    g_proportion_not_contained = proportion_not_contained(target, ref_lb, ref_ub),
    g_empirical_cdf = empirical_cdf(year, target, ref, sample, slope),
    g_t_test = t_test(year, target, ref, sample, slope),
    # auxiliary inputs to normalization
    min_envelope = min(ref_lb),
    max_envelope = max(ref_ub)
  ) %>%
  ungroup() %>%
  mutate(
    # normalization implied by axis scale (range from no slope to aspect ratio of chart)
    g_axis_normalized_difference = normalize(g_difference, 0, aspect_ratio),
    # normalization implied by signals in envelope (range from no slope to max slope in
 envelope)
    g_envelope_normalized_difference = normalize(g_difference, 0, (max_envelope - min_en
velope) / chart_width_pixels),
    # normalization implied by observed signals
    g_difference = normalize(g_difference, min(g_difference), max(g_difference)),
    g_proportion_not_contained = normalize(g_proportion_not_contained, min(g_proportion_
not_contained), max(g_proportion_not_contained)),
    g_empirical_cdf = normalize(g_empirical_cdf, min(g_empirical_cdf), max(g_empirical_c
df)),
    g_t_test = normalize(g_t_test, min(g_t_test), max(g_t_test))
  ) %>%
  pivot_longer(
    cols = starts_with("g_"),
    names_to = "discrepancy_function",
    names_prefix = "g_",
    values_to = "signal",
    values_drop_na = TRUE
  ) %>%
  unite("scale", aspect_ratio:axis_limits, remove = FALSE) %>%
  ggplot(aes(x = target_slope_diff, y = signal, color = discrepancy_function)) +
  geom_line() +
  colScale +
```

```
  theme_minimal() +
  facet_grid(sd ~ scale)
```

```
## `summarise()` regrouping output by 'target_slope_diff', 'sd', 'aspect_ratio', 'axis_l
imits' (override with `.groups` argument)
```



Since we are now estimating the reference average and envelope based on an ensemble, all of the discrepancy functions make slightly different predictions, although the patterns are recognizably similar to before. Examining the two new discrepancy functions we've added, we can see that the empirical cdf overestimates signal relative to a visual t-test and that for all but the lowest level of variance, alternatives to the visual t-test are not sufficiently conservative about the amount of signal in a chart.

NOTE: Although I've added a way to manipulate the `sample` size used for the empirical cdf and t-test discrepancy functions in the code block above (currently n = 20), I think we could still get more detailed about how this sampling might happen differently for a static ensemble versus HOPs. For example, when the whole ensemble is visible at once, people may be more likely to sample the boundaries of the implied envelope of uncertainty. Whereas for HOPs, the samples arrive in `draw` order and the user samples however many of them they attend to, including for the purpose of ensemble processing which isn't yet reflected in the calculations above.