The Q-Q points are easy to calculate

Fox's regression textbook (2016, p. 38–39) provides a step-by-step procedure for building a Q-Q plot and includes a great implementation in the book's companion R package with `car::qqPlot()`. His procedure will be the basis for the math in this post

The first step is to convert ranks into quantiles. One naive approach might be dividing ranks by the length:

```
# generate data for this post
set.seed(20200825)
x <- sort(rnorm(20, 10, 3))

# naive quantiles
rank(x) / length(x)
#>  [1] 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70
0.75
#> [16] 0.80 0.85 0.90 0.95 1.00
```

With 20 observations, 1 point is 5% of the data, so the quantiles increase by .05 with each step from .05 to 1.00. The problem here is that we get a quantile of 1. When we look up the quantiles from theoretical distributions, quantiles of 0 and 1 break things:[1]

```
# 0 and 1 quantiles of a normal distribution
qnorm(c(0, 1))
#> [1] -Inf  Inf
```

Instead, we will adjust the ranks by 1/2, so that the quantiles are "centered" in their .05 bins. That is, the first quantile of .05 will become .025.

```
q <- (rank(x) - .5) / length(x)
q
#>  [1] 0.025 0.075 0.125 0.175 0.225 0.275 0.325 0.375 0.425 0.475 0.525 0.575
#> [13] 0.625 0.675 0.725 0.775 0.825 0.875 0.925 0.975
```

R provides a function for this purpose, `ppoints()`, but it includes extra smarts for when there are 10 or fewer values. Otherwise, it does the same adjustment by .5 that we are using.

```
ppoints
#> function (n, a = if (n <= 10) 3/8 else 1/2)
#> {
#>     if (length(n) > 1L)
#>         n <- length(n)
#>     if (n > 0)
#>         (1L:n - a)/(n + 1 - 2 * a)
#>     else numeric()
#> }
#>
#>
```

Given these quantiles, we can look up which values have these quantiles in various distributions. For example, for a normal distribution with a mean of 100 and standard deviation of 15, the corresponding quantiles would be:

```
round(qnorm(q, 100, 15))
```

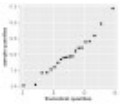
```
#>  [1]  71  78  83  86  89  91  93  95  97  99 101 103 105 107 109 111 114 117
122
#> [20] 129
```

Or, more to the point, we can project our observed data onto a normal distribution with the same mean and standard deviation as our data.

```
qnorm(q, mean(x), sd(x))
#>  [1]  4.087038  5.639679  6.502415  7.146107  7.680649  8.150992  8.580590
#>  [8]  8.983711  9.370120  9.747252 10.121407 10.498539 10.884947 11.288068
#> [15] 11.717667 12.188009 12.722552 13.366243 14.228980 15.781621
```

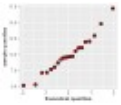
Let's bundle things into a dataframe and start making some plots.

```
library(tibble)
library(ggplot2)

d <- tibble(
  x_sample = x,
  quantile = ppoints(length(x_sample)),
  x_theoretical = qnorm(q, mean(x_sample), sd(x_sample))
)

ggplot(d) +
  geom_point(aes(x = x_theoretical, y = x_sample)) +
  labs(
    x = "theoretical quantiles",
    y = "sample quantiles"
  )
```



If we project onto the *z*-score distribution—that is, a "unit normal", a normal with mean 0 and standard deviation 1—our points (black) match the Q-Q plot provided by ggplot2 (red points):

```
d$z_theoretical <- qnorm(d$quantile, 0, 1)

ggplot(d) +
  geom_point(aes(x = z_theoretical, y = x_sample), size = 3) +
  geom_qq(aes(sample = x_sample), color = "red") +
  labs(
    x = "theoretical quantiles",
    y = "sample quantiles"
  )
```



A math-less description of what we are doing

We can think of this quantile-quantile procedure in terms of database operations:

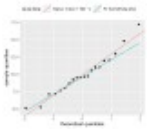
- Start with two tables `a` and `b` with the same numbers of rows
- For each one, create a `quantile` "index" for the value of interest.
- Now join the datasets using the quantile index: `left_join(a, b, by = "quantile")`

If `a` and `b` were tables of real-life collected data, that's the procedure we would follow. For Q-Q plots, however, the second dataset `b` almost always isn't a table of observations. Instead, we "look up" of the values using math by computing values from a theoretical distribution. (Or in an older stats textbook, you actually would look up these values in a table in the back of the book.)

The Q-Q line is trickier

Q-Q plots usually come with a reference line so that we can assess whether the two sets of quantiles follow a linear trend. Until recently, I thought the line was \(y = \mathrm{mean} + \mathrm{SD} * x\), so that the line increased by one standard deviation on the sample scale (*y*) given a one standard deviation change in the theoretical scale (*x*). But that is not the case:

```
p <- ggplot(d) +
  geom_point(aes(x = z_theoretical, y = x_sample)) +
  geom_abline(
    aes(intercept = mean, slope = sd, color = "Naive: mean + SD * x"),
    data = tibble(sd = sd(d$x_sample), mean = mean(d$x_sample))
  ) +
  geom_qq_line(
    aes(sample = x_sample, color = "R: Something else"),
    # use the abline glyph in the legend
    key_glyph = draw_key_abline
  ) +
  labs(
    color = "Q-Q line",
    x = "theoretical quantiles",
    y = "sample quantiles"
  ) +
  guides(color = guide_legend(nrow = 1)) +
  theme(legend.position = "top", legend.justification =  "left")
p
```



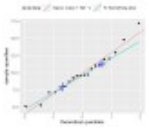
This recent discovery mystified me. This line from the help page for `qqline()` offered a clue:

> `qqline` adds a line to a "theoretical", by default normal, quantile-quantile plot which passes through the probs quantiles, by default the first and third quartiles.

Indeed, if we draw points at the .25 and .75 quantiles, we can see that the they land on R's Q-Q line.

```
anchors <- tibble(
  x = qnorm(c(.25, .75)),
  y = quantile(d$x_sample, c(.25, .75))
)

p +
  geom_point(
    aes(x = x, y = y),
    data = anchors,
    shape = 3,
```

```
    size = 5,
    stroke = 1.1,
    color = "blue"
  )
```

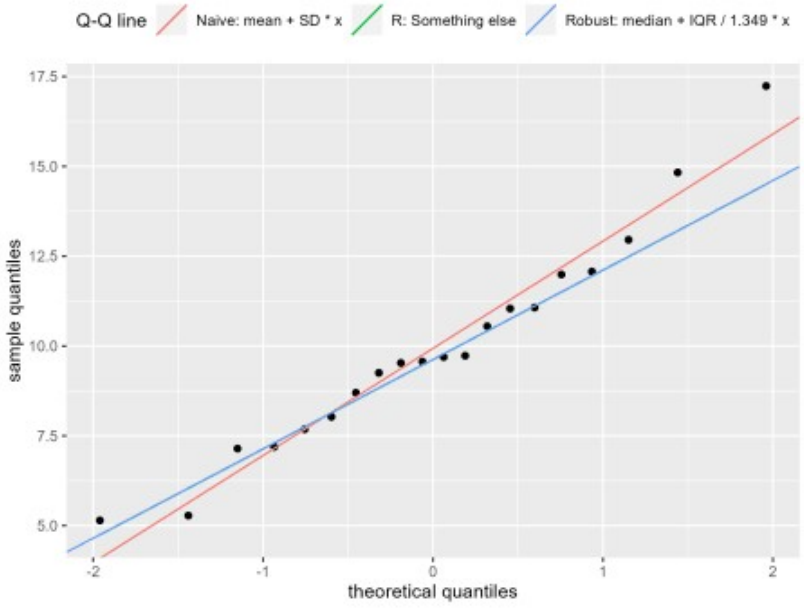


Fox explains that this is a “robust” estimate for the line:

> We can alternatively use the median as a robust estimator of [the mean] and the interquartile range / 1.349 as a robust estimator of [the standard deviation]. (The more conventional estimates [of the sample mean and SD] will not work well when the data are substantially non-normal.) [p. 39].

We can confirm the previous plot by updating the previous plot. We draw a third line for the robust estimate and it follows the line from `geom_qqline()`.

```
p +
  geom_abline(
    aes(
      intercept = mean,
      slope = sd,
      color = "Robust: median + IQR / 1.349 * x"
    ),
    data = tibble(
      sd = IQR(d$x_sample) / 1.349,
      mean = median(d$x_sample)
    )
  )
```



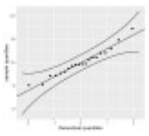
A confidence band helps

One problem with these Q-Q plots is that it is hard to tell whether the points are straying too far away from the line. We can include a 95% confidence band to support interpretation. Fox provides an equation for the standard error for a quantile. It appears to be the square root of the variance of a quantile. The function `se_z()` applies the equation to a *z*-score (the theoretical values on the *x* axis).

```
# Given z scores and n, produce a standard error
se_z <- function(z, n) {
  sqrt(pnorm(z) * (1 - pnorm(z)) / n) / dnorm(z)
}
```

In the code below, we create a grid of 300 *z* values and compute the standard error at those values. We weight the standard errors by the standard deviation to convert to the sample scale and multiply the standard errors by 2 and -2 to get approximate 95% intervals around the reference lines. We do this whole procedure again but using robust estimates.

```
band <- tibble(
  z = seq(-2.2, 2.2, length.out = 300),
  n = length(d$x_sample),

  sample_sd = sd(d$x_sample),
  se = sample_sd * se_z(z, n),
  line = mean(d$x_sample) + sample_sd * z,
  upper = line + 2 * se,
  lower = line - 2 * se,

  robust_sd = IQR(d$x_sample) / 1.349,
  robust_line = median(d$x_sample) + z * robust_sd,
  robust_se =  robust_sd * se_z(z, n),
  robust_upper = robust_line + 2 * robust_se,
  robust_lower = robust_line - 2 * robust_se,
)

ggplot(d) +
  geom_point(aes(x = z_theoretical, y = x_sample)) +
  geom_abline(
    aes(intercept = mean, slope = sd, color = "Naive"),
    data = tibble(sd = sd(d$x_sample), mean = mean(d$x_sample))
  ) +
  geom_ribbon(
    aes(x = z, ymax = upper, ymin = lower, color = "Naive"),
    data = band,
    fill = NA, show.legend = FALSE
  ) +
  labs(
    color = "Q-Q line",
    x = "theoretical quantiles",
    y = "sample quantiles"
  ) +
  guides(color = guide_legend(nrow = 1)) +
  theme(legend.position = "top", legend.justification =  "left")
```

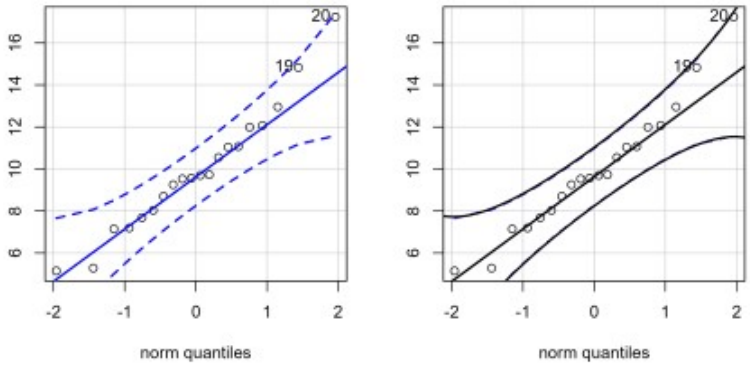


We would like to see these points track along the reference line, but the confidence band shows that we can expect the quantiles in the tails will be a little noisier than the ones in the middle. Fox notes that these confidence values are *pointwise* (applying to individual points) and not *simultaneous* (applying to the whole band), meaning there is "there is a greater probability that *at least point* point strays outside the envelope even if the data are sampled from the comparison distribution. [p. 39, footnote]". (I am not quite

sure what that means, honestly.)

Let's also check our work against the car package which provides robust confidence bands with its Q-Q plots. We first plot the car package's Q-Q plot and observe its confidence band in blue. Then we plot it again but draw black lines for our hand-calculated confidence band. Because our lines overlap the car package's lines, we essentially match its implementation. (Internally, it uses 1.96 standard errors. We used 2 standard errors.)

```
# Set margins on Q-Q plots
par(mar = c(4, 2, 1, 2))

# Use patchwork to capture the plots and combine them
# into a side by side display.
library(patchwork)
p1 <- wrap_elements(~ car::qqPlot(x))

p2 <- wrap_elements(~ {
  car::qqPlot(x)
  lines(band$z, band$robust_line, col = "black", lwd = 2)
  lines(band$z, band$robust_upper, col = "black", lwd = 2)
  lines(band$z, band$robust_lower, col = "black", lwd = 2)
})

p1 + p2
```



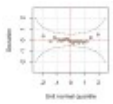
Worm plots

Finally, I would like to implement my preferred alternative to Q-Q plot: the worm plot. I first encountered these in the gamlss package. The idea is appealing: The Q-Q plot wastes a lot of vertical space showing a line go upwards. If we remove that line—or rotate it to be a horizontal line—then we can devote the vertical space in the plot for showing the deviation around the reference line. These are sometimes called "detrended" Q-Q plots because they remove the diagonal trend of the Q-Q comparison line.

We can use the gamlss implementation if we pretend that the data are residuals from a model:

```
par(mar = c(4.5, 4.5, 1, 2))

# I don't know what's up with that error message.

# use scale() to transform in to z-score
gamlss::wp(
```

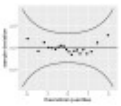
```
  resid = scale(d$x_sample),
  xlim.all = 2.5,
  line = FALSE
)
#> Error in coef(fit): object 'fit' not found
```



This plot shows the basic idea: We are comparing quantiles along a line and we have a confidence band that helps us gauge the deviation around that line.

Because the “trend” we are going to “detrend” is just a reference line, we can detrend the data by subtracting the line. Here we use the “naive” calculation of the line for a worm plot.

```
d$line <- mean(d$x_sample) + d$z_theoretical * sd(d$x_sample)

ggplot(d) +
  geom_point(
    aes(x = z_theoretical, y = x_sample - line)
  ) +
  geom_hline(yintercept = 0) +
  geom_ribbon(
    # don't add the line to the SEs
    aes(x = z, ymax = 2 * se, ymin = - 2 * se),
    data = band,
    fill = NA,
    color = "black"
  ) +
  labs(
    color = "Q-Q line",
    x = "theoretical quantiles",
    y = "centered sample quantiles"
  ) +
  guides(color = guide_legend(nrow = 1)) +
  theme(legend.position = "top", legend.justification =  "left")
```



The values on the *y* axis differ between this plot and the one from `wp()`. That’s because the data for `wp()` were fully rescaled with `scale()`: They were mean centered and divided by the standard deviation to become *z* scores. Here we only subtracted the mean line.

Q-Q Recap

That covers the main points I’d like to make about Q-Q plots:

- We are joining two distributions together using quantiles and comparing them visually.
- The default Q-Q plot uses robust estimates of the mean and standard deviation.
- `car::qqPlot()` provides the best option for routine visualization.
- Worm plots subtract the reference line from the points, so I think they are better option.

Bonus: Let's create `stat_worm()`

I would like to implement a basic normal-family worm plot in ggplot2 so that I can call `stat_worm()` and `stat_worm_band()`. I would also like to learn how to write my own extensions to ggplot2, so this task is useful learning exercise. I admit that I copied extensively from some sources:

- The source code for `stat_qq()`
- The extensions vignette for ggplot2
- The extensions chapters in the ggplot2 book

The code walkthrough below will make the work seem effortless, but I spent a while trying to figure it out!

The apparent workflow is to write a user-facing "layer" function and then create `ggProto()` objects that do the work behind the scenes, so that's what we will do.

We start with `stat_worm()`. I copied the `stat_qq()` source code and removed some options for supporting other distributions. I code-commented above the lines that are not boilerplot.

```
stat_worm <- function(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  # important part
  robust = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
) {
  layer(
    data = data,
    mapping = mapping,
    # important part
    stat = StatWorm,
    geom = geom,
    position = position,
    show.legend = show.legend,
    inherit.aes = inherit.aes,
    # important part
    params = list(
      robust = robust,
      na.rm = na.rm,
      ...
    ),
  )
}
```

The main work will be carried out by `stat = StatWorm`, the `ggProto` object that will to the calculations. The computed values will be

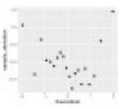
drawn, by default, with `geom = "point"`. I provide an option for `robust` estimates in the function arguments and in the list of `params`.

Now we create `StatWorm`. The `default_aes` and `required_aes` set up the aesthetic mappings. The main work is in `compute_group()`. It will receive a dataframe `data` with a column of the `sample` values, and we compute naive or robust quantile values as above. The function returns an updated dataframe with the columns `sample` and `theoretical` which match the `after_stat()` calls in `default_aes`.

```
StatWorm <- ggproto(
  "StatWorm",
  Stat,
  default_aes = aes(
    y = after_stat(sample),
    x = after_stat(theoretical)
  ),
  required_aes = c("sample"),
  compute_group = function(data, scales, robust = FALSE) {
    sample <- sort(data$sample)
    n <- length(sample)
    quantiles <- ppoints(n)

    if (robust) {
      mean <- median(sample)
      sd <- IQR(sample) / 1.349
    } else {
      mean <- mean(sample)
      sd <- sd(sample)
    }

    scaled_theoretical <- qnorm(quantiles, mean, sd)
    theoretical <- qnorm(quantiles)

    data.frame(
      # detrended
      sample = sample - scaled_theoretical,
      theoretical = theoretical
    )
  }
)
```

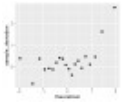
Let's give it a try:

```
ggplot(d) +
  stat_worm(aes(sample = x_sample), robust = FALSE) +
  # test against above code
  geom_point(aes(x = z_theoretical, y = x_sample - line)) +
  labs(y = "centered sample")
```



```
d$robust_line <-
  median(d$x_sample) + d$z_theoretical * IQR(d$x_sample) / 1.349

ggplot(d) +
```

```
  stat_worm(aes(sample = x_sample), robust = TRUE) +
  # test against above code
  geom_point(aes(x = z_theoretical, y = x_sample - robust_line)) +
  labs(y = "centered sample")
```



Now, we work through the same procedure with the confidence band. First, create the user-facing layer function that we want to use.

```
stat_worm_band <- function(
  mapping = NULL,
  data = NULL,
  geom = "ribbon",
  position = "identity",
  ...,
  # important part
  robust = FALSE,
  band_width = .95,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
) {
  layer(
    data = data,
    mapping = mapping,
    # important part
    stat = StatWormBand,
    geom = GeomRibbonHollow,
    position = position,
    show.legend = show.legend,
    inherit.aes = inherit.aes,
    # important part
    params = list(
      robust = robust,
      band_width = band_width,
      na.rm = na.rm,
      ...
    ),
  )
}
```

The easiest way to draw two lines in a confidence band is with `geom_ribbon()`, but `geom_ribbon()` by default draws a filled shape. It would fill the space between the lines with solid black. (It was pain while developing this code.) Fortunately, the vignette gives an example of modifying an existing geom to take new defaults. We follow that example and create a new `GeomRibbonHollow`:

```
GeomRibbonHollow <- ggproto(
  # Make this geom
  "GeomRibbonHollow",
  # inheriting from:
  GeomRibbon,
  # but changing this:
  default_aes = aes(
    colour = "black",
    fill = NA,
```

```
    size = 0.5,
    linetype = 1,
    alpha = NA)
  )
```

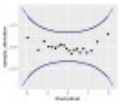
Next, we create `StatWormBand`. It's the same kind of code as above, just operating generically on a dataframe called `data` with a column called `sample`.

```
StatWormBand <- ggproto(
  "StatWormBand",
  Stat,
  default_aes = aes(
    x = after_stat(theoretical),
  ),
  required_aes = c("sample"),
  compute_group = function(data, scales, robust = FALSE, band_width = .95) {
    sample <- sort(data$sample)
    n <- length(sample)
    quantiles <- ppoints(n)

    if (robust) {
      mean <- median(sample)
      sd <- IQR(sample) / 1.349
    } else {
      mean <- mean(sample)
      sd <- sd(sample)
    }

    theoretical <- qnorm(quantiles)
    z_range <- seq(min(theoretical), max(theoretical), length.out = 80)

    # i.e., convert .95 to .025 and .975 and convert those to z scores
    band_z <- qnorm((1 + c(-band_width, band_width)) / 2)

    se_z <- function(z, n) {
      sqrt(pnorm(z) * (1 - pnorm(z)) / n) / dnorm(z)
    }

    data.frame(
      theoretical = z_range,
      ymin = band_z[1] * se_z(z_range, n) * sd,
      ymax = band_z[2] * se_z(z_range, n) * sd
    )
  }
)
```

Okay, moment of truth:

```
ggplot(d) +
  stat_worm(aes(sample = x_sample)) +
  # original version from above
  geom_ribbon(
    aes(x = z, ymax = upper - line, ymin = lower - line),
    data = band,
    fill = NA,
    color = "blue"
  ) +
```

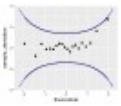
```
  stat_worm_band(aes(sample = x_sample)) +
  labs(y = "centered sample")
```



The bands differ slightly because `stat_worm_band()` uses 1.96 standard errors for its confidence band (instead of 2) and because `stat_worm_band()` matches the range of the data (our earlier example manually set the range).

Finally, let's look at the robust version:

```
ggplot(d) +
  stat_worm(aes(sample = x_sample), robust = TRUE) +
  # for comparison
  geom_ribbon(
    aes(
      x = z,
      ymax = robust_upper - robust_line,
      ymin = robust_lower - robust_line
    ),
    data = band,
    fill = NA,
    color = "blue"
  ) +
  stat_worm_band(aes(sample = x_sample), robust = TRUE) +
  labs(y = "centered sample")
```



This is a decent start. A more complete version would support other reference distributions besides the normal distribution and allow some calculations to be fixed. That is, with this implementation, if we facet or color the lines, the statistics are calculated separately for each facet, color, etc. Each subgroup of data is a new dataset, not a highlighted part of the original distribution. Plotting different subsets of data in different colors, but keeping them part of the same quantile calculation, would be useful for exploring which subsets of data are deviating away from 0.

1.