# Effect of Scaling on Colr

Mike Krähenbühl

2025-04-03

## Impact of Scaling (simulated data)

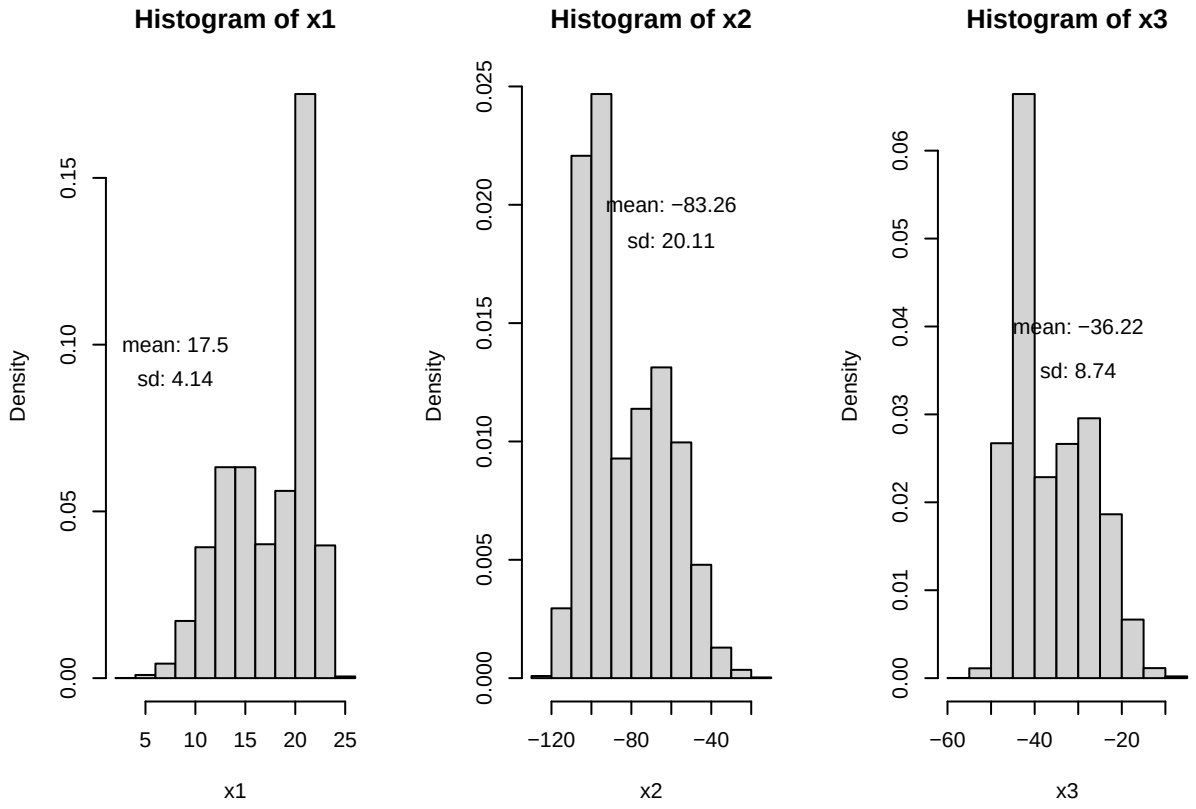
### Scaling simulation

Simulate X1, X2 and X3 with a linear transformation function and check the impact of scaling on the coefficients.

```
set.seed(123)
n <- 10000

X_1_A = rnorm(n, 14, 3)
X_1_B = rnorm(n, 21, 1)
x1 = ifelse(sample(1:2, replace = TRUE, size = n) == 1, X_1_A, X_1_B)


U2 = runif(n)
x_2_dash = qlogis(U2)

#x_2_dash = h_0(x_2) + beta * X_1
#x_2_dash = 0.42 * x_2 + 2 * X_1

x2 = 1/0.42 * (x_2_dash - 2 * x1)


U3 = runif(n)
x_3_dash = qlogis(U3)

#x_3_dash = h_0(x_3) + beta * X_2
#x_3_dash = 3 * x_2 + 0.5 * X_1 - 1.2 * X_2

x3 = 1/3 * (x_3_dash - 0.5 * x1 + 1.2 * x2)
```

```
par(mfrow=c(1,3))

hist(x1, freq = FALSE)
# add mean and sd of x1 to plot as text
text(8, 0.1, paste("mean:", round(mean(x1), 2)))
text(8, 0.09, paste("sd:", round(sd(x1), 2)))

hist(x2, freq = FALSE)
```

```r
# add mean and sd of x2 to plot as text
text(-60, 0.02, paste("mean:", round(mean(x2), 2)))
text(-60, 0.0185, paste("sd:", round(sd(x2), 2)))

hist(x3, freq = FALSE)
# add mean and sd of x3 to plot as text
text(-30, 0.04, paste("mean:", round(mean(x3), 2)))
text(-30, 0.035, paste("sd:", round(sd(x3), 2)))
```




Summary table of the 3 variables raw and standardized:

```r
x1_std <- (x1 - mean(x1)) / sd(x1)
x2_std <- (x2 - mean(x2)) / sd(x2)
x3_std <- (x3 - mean(x3)) / sd(x3)

# summary table

summary_table <- data.frame(
  Variable = c("x1", "x2", "x3"),
  Mean = c(mean(x1), mean(x2), mean(x3)),
  SD = c(sd(x1), sd(x2), sd(x3)),
  Mean_std = c(mean(x1_std), mean(x2_std), mean(x3_std)),
  SD_std = c(sd(x1_std), sd(x2_std), sd(x3_std))
)

# round numeric values
```

```r
summary_table[, 2:5] <- round(summary_table[, 2:5], 2)


kable(summary_table, caption = "Summary table of the 3 variables raw and standardized")
```

Table 1: Summary table of the 3 variables raw and standardized

| Variable | Mean | SD | Mean_std | SD_std |
|---|---|---|---|---|
| x1 | 17.50 | 4.14 | 0 | 1 |
| x2 | -83.26 | 20.11 | 0 | 1 |
| x3 | -36.22 | 8.74 | 0 | 1 |

**Coefficients change when scaling**

Linear Regression:

Fit Lm() with raw or standardized predictors or outcome.

```r
lm_raw <- lm(x3 ~ x1 + x2)
lm_std <- lm(x3 ~ x1_std + x2_std)
lm_all_std <- lm(x3_std ~ x1_std + x2_std)

round(coef(lm_raw), 5)
```

```
## (Intercept)          x1          x2
##    -0.00268    -0.15514     0.40241
```

```r
round(coef(lm_std), 5)
```

```
## (Intercept)      x1_std      x2_std
##   -36.22252    -0.64225     8.09419
```

```r
round(coef(lm_all_std), 5)
```

```
## (Intercept)      x1_std      x2_std
##     0.00000    -0.07346     0.92576
```

Coefficients of the model with the raw outcome and standardized predictors lm(x3 ~ x1_std + x2_std) are the same as the coefficients of the all standardized model lm(x3_std ~ x1_std + x2_std) multiplied by the standard deviation of the raw outcome. Makes sense, because the models imply a linear shift in the expected raw or standardized value for a 1 unit increase in standardized predictors.

```r
round(coef(lm_all_std)*sd(x3), 5)
```

```
## (Intercept)      x1_std      x2_std
##     0.00000    -0.64225     8.09419
```

The coefficients of the standardized predictor model lm(x3 ~ x1_std + x2_std) are equal to the predictors in the raw predictor model lm(x3 ~ x1 + x2) times the standard deviations of the respective predictors.

```
paste0("sd(x1): ", round(sd(x1), 5),
       ", sd(x2): ", round(sd(x2), 5))
```

```
## [1] "sd(x1): 4.13968, sd(x2): 20.11432"
```

```
round(coef(lm_std)/coef(lm_raw), 5)
```

```
## (Intercept)      x1_std      x2_std
## 13529.86104     4.13968    20.11432
```

Now same analysis for the Colr model:

The coefficients for the the models with standardized predictors and raw or standardized outcomes are equal. The reason is, that the log-odds are scale invariant. It does not matter if the outcome is standardized or not.

```
colr_raw <- Colr(x3 ~ x1 + x2)
colr_std <- Colr(x3 ~ x1_std + x2_std)
colr_all_std <- Colr(x3_std ~ x1_std + x2_std)

round(coef(colr_raw), 5)
```

```
##       x1       x2
##  0.47559 -1.21093
```

```
round(coef(colr_std), 5)
```

```
##    x1_std    x2_std
##   1.96897 -24.35713
```

```
round(coef(colr_all_std), 5)
```

```
##    x1_std    x2_std
##   1.96897 -24.35713
```

The coefficients of the standardized predictor model should be equal to the predictors in the raw predictor model times the standard deviations of the respective predictors. However, there is a small difference. Maybe numerical issue? –> in a linear model, this is exact. so could the issue be that in the Colr model we model a smooth transformation function and scaling (btw. subtracting the mean), changes the reference configuration and therefore lead to another smooth transformation function?

```
paste0("sd(x1): ", round(sd(x1), 5),
       ", sd(x2): ", round(sd(x2), 5))
```

```
## [1] "sd(x1): 4.13968, sd(x2): 20.11432"
```

```
round(coef(colr_std)/coef(colr_raw), 5)
```

```
##   x1_std   x2_std
##  4.14010 20.11435
```

### Analyze Colr() when scaling

Analyze Colr(x2 ~ x1, order=len_theta):

```
len_theta <- 20

fit_all_raw <- Colr(x2 ~ x1, order=len_theta)
fit_all_std <- Colr(x2_std ~ x1_std, order=len_theta)
fit_x2_raw <- Colr(x2 ~ x1_std, order=len_theta)
fit_x2_std <- Colr(x2_std ~ x1, order=len_theta)
```

```
coef(fit_all_raw)
```

```
##       x1
## 1.989529
```

```
coef(fit_all_std)
```

```
##   x1_std
## 8.237265
```

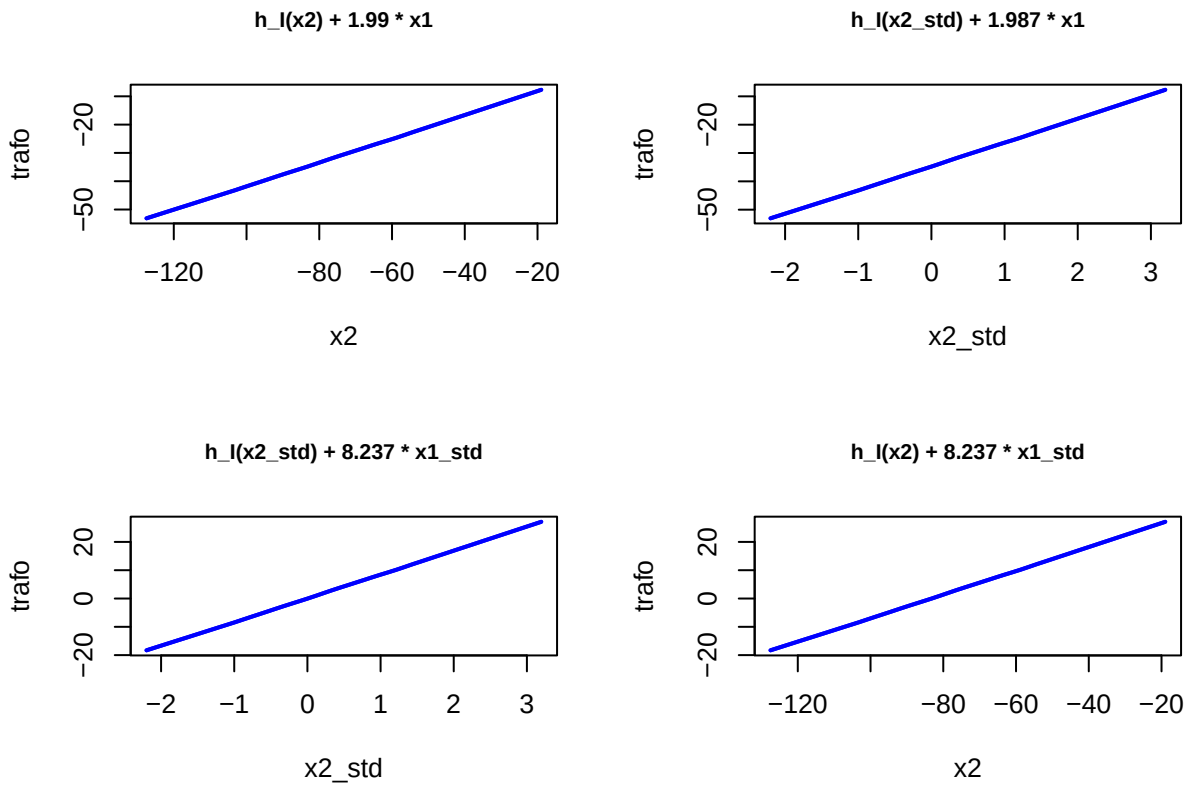
```
coef(fit_x2_raw)
```

```
##   x1_std
## 8.237102
```

```
coef(fit_x2_std)
```

```
##       x1
## 1.986779
```

```
# plot baseline trafo

par(mfrow=c(2,2))
temp = model.frame(fit_all_raw)[1:2,-1, drop=FALSE]
plot(fit_all_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2) + ", round(coef(fit_all_raw)[1], 3), " * x1"), cex.main=0.8)
temp = model.frame(fit_x2_std)[1:2,-1, drop=FALSE]
plot(fit_x2_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2_std) + ", round(coef(fit_x2_std)[1], 3), " * x1"), cex.main=0.8)
temp = model.frame(fit_all_std)[1:2,-1, drop=FALSE]
plot(fit_all_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2_std) + ", round(coef(fit_all_std)[1], 3), " * x1_std"), cex.main=0.8)
temp = model.frame(fit_x2_raw)[1:2,-1, drop=FALSE]
plot(fit_x2_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2) + ", round(coef(fit_x2_raw)[1], 3), " * x1_std"), cex.main=0.8)
```




The coefficients change if we use raw or scaled predictors. It does not matter though if we use the raw or scaled outcome. The coefficients are the same.

Analyze Colr(x3 ~ x1 + x2, order=len_theta) :

```
fit_all_raw <- Colr(x3 ~ x1 + x2, order=len_theta)
fit_all_std <- Colr(x3_std ~ x1_std + x2_std, order=len_theta)
fit_x3_raw <- Colr(x3 ~ x1_std + x2_std, order=len_theta)
fit_x3_std <- Colr(x3_std ~ x1 + x2, order=len_theta)
```

```
coef(fit_all_raw)
```

```
##         x1         x2
##  0.4694353 -1.1972386
```

```
coef(fit_all_std)
```

```
##     x1_std     x2_std
##   1.976529 -24.362158
```

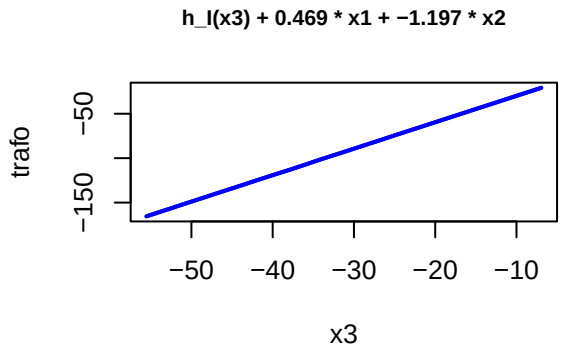
```
coef(fit_x3_raw)
```

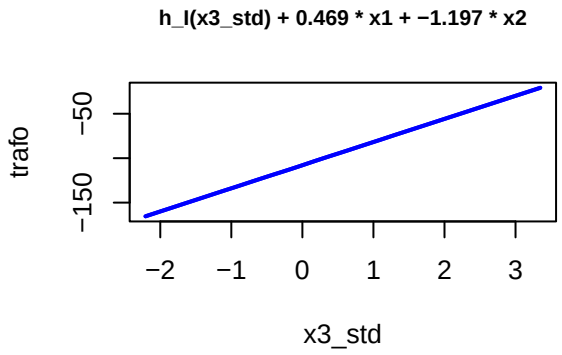
```
##     x1_std     x2_std
##   1.975725 -24.358114
```

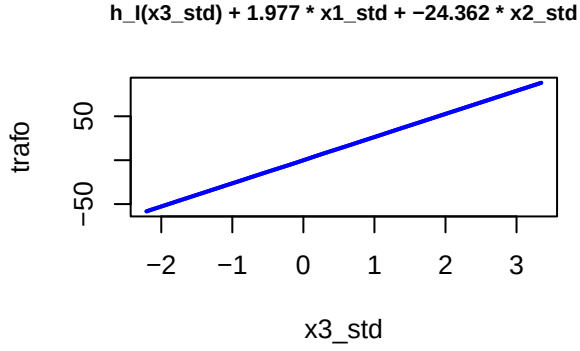
```
coef(fit_x3_std)
```

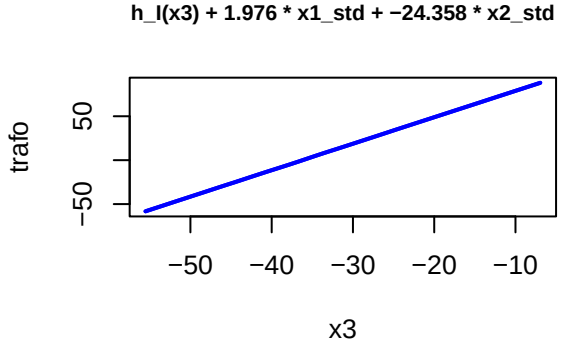
```
##         x1         x2
##  0.4693279 -1.1967814
```

```
# plot baseline trafo
par(mfrow=c(2,2))
temp = model.frame(fit_all_raw)[1:2,-1, drop=FALSE]
plot(fit_all_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3) + ", round(coef(fit_all_raw)[1], 3),
                 " * x1 + ", round(coef(fit_all_raw)[2], 3), " * x2"), cex.main=0.8)
temp = model.frame(fit_x3_std)[1:2,-1, drop=FALSE]
plot(fit_x3_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3_std) + ", round(coef(fit_x3_std)[1], 3),
                 " * x1 + ", round(coef(fit_x3_std)[2], 3), " * x2"), cex.main=0.8)

temp = model.frame(fit_all_std)[1:2,-1, drop=FALSE]
plot(fit_all_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3_std) + ", round(coef(fit_all_std)[1], 3),
                 " * x1_std + ", round(coef(fit_all_std)[2], 3), " * x2_std"), cex.main=0.8)

temp = model.frame(fit_x3_raw)[1:2,-1, drop=FALSE]
plot(fit_x3_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3) + ", round(coef(fit_x3_raw)[1], 3),
                 " * x1_std + ", round(coef(fit_x3_raw)[2], 3), " * x2_std"), cex.main=0.8)
```



Check how the coefficients change:

The coefficient for the standardized predictor is approx. equal to the coefficient for the raw predictor multiplied by the standard deviation of the raw predictor.

```
# beta1 for x1
paste0("raw x1: beta_raw = ", round(coef(fit_all_raw)[1], 3))
```

```
## [1] "raw x1: beta_raw = 0.469"
```

```
paste0("std x1: beta_std = ", round(coef(fit_x3_raw)[1], 3))
```

```
## [1] "std x1: beta_std = 1.976"
```

```
paste0( "sd(x1) = ", round(sd(x1), 3))
```

```
## [1] "sd(x1) = 4.14"
```

```
paste0("beta_raw * sd(x1)  =  ", round(coef(fit_all_raw)[1]*sd(x1), 3))
```

```
## [1] "beta_raw * sd(x1)  =  1.943"
```

```
paste0(" ")
```

```
## [1] " "
```

```
# same for x2
paste0("raw x2: beta_raw = ", round(coef(fit_all_raw)[2], 3))
```

```
## [1] "raw x2: beta_raw = -1.197"
```

```
paste0("std x2: beta_std = ", round(coef(fit_x3_raw)[2], 3))
```

```
## [1] "std x2: beta_std = -24.358"
```

```
paste0( "sd(x2) = ", round(sd(x2), 3))
```

```
## [1] "sd(x2) = 20.114"
```

```
paste0("beta_raw * sd(x2)  =  ", round(coef(fit_all_raw)[2]*sd(x2), 3))
```

```
## [1] "beta_raw * sd(x2)  =  -24.082"
```

We standardize the outcome $Y$ as follows:

$$Y_{\text{std}} = \frac{Y - \mu_Y}{\sigma_Y}$$

This transformation is linear, monotonic, and invertible:

$$Y = Y_{\text{std}} \cdot \sigma_Y + \mu_Y$$

Therefore, for any threshold $y$, we have the equivalence:

$$P(Y < y \mid X) = P\left(Y_{\text{std}} < \frac{y - \mu_Y}{\sigma_Y} \mid X\right)$$

This means that the probability is the identical when evaluating the same quantile in the standardized outcome as in the raw outcome. Furthermore, the interpretation of coefficients in a cumulative odds logistic regression remains unchanged.

In particular, the log-odds ratio:

$$\log\left(\frac{P(Y < y \mid X + 1)}{1 - P(Y < y \mid X + 1)}\right) - \log\left(\frac{P(Y < y \mid X)}{1 - P(Y < y \mid X)}\right)$$

is equal to:

$$\log\left(\frac{P\left(Y_{\text{std}} < \frac{y-\mu_Y}{\sigma_Y} \mid X + 1\right)}{1 - P\left(Y_{\text{std}} < \frac{y-\mu_Y}{\sigma_Y} \mid X + 1\right)}\right) - \log\left(\frac{P\left(Y_{\text{std}} < \frac{y-\mu_Y}{\sigma_Y} \mid X\right)}{1 - P\left(Y_{\text{std}} < \frac{y-\mu_Y}{\sigma_Y} \mid X\right)}\right)$$

as long as the same quantile (i.e. probability threshold) is used. Thus, the coefficient $\beta$ reflects the same change in log-odds for a one-unit increase in the standardized predictor, even if the outcome is also standardized.

For example, if we want to know the probability $P(Y < 20 \mid X = 3)$:

General model formula:

$$P(Y < y \mid X = x) = F_z\left(h(Y) + \beta \cdot X\right)$$

But our Model is fitted like this:

$$P(Y_{\text{std}} < y_{\text{std}} \mid X_{\text{std}} = x_{\text{std}}) = F_z\left(h(Y_{\text{std}}) + \beta' \cdot X_{\text{std}}\right)$$

$$P\left(\frac{Y - \mu_Y}{\sigma_Y} < \frac{20 - \mu_Y}{\sigma_Y} \,\middle|\, X_{\text{std}} = \frac{3 - \mu_X}{\sigma_X}\right) = F_z\left(h\left(\frac{20 - \mu_Y}{\sigma_Y}\right) + \beta' \cdot \frac{3 - \mu_X}{\sigma_X}\right)$$

$P(X3 < (-18) \mid X1 = 15, X2 = -40)$ on the DGP:

```
# x3 = 1/3 * (x_3_dash - 0.5 * x1 + 1.2 * x2)

# plug in the 3 variables
# -18 = 1/3 * (x_3_dash - 0.5 * 15 + 1.2 * -40)

# solve for x_3_dash (log-odds)
x_3_dash <- -18 * 3 + 0.5 * 15 - 1.2 * -40

# get probability
plogis(x_3_dash)
```

```
## [1] 0.8175745
```

$P(X3 < (-18) \mid X1 = 15, X2 = -40)$ on the raw data:

```
fit_all_raw <- Colr(x3 ~ x1 + x2, order=len_theta)

# for x1 = 15, x2 = -40
# outcome x3 < -18

newdata <- data.frame(x1 = 15, x2 = -40, x3 = -18)
pred_log_odds <- predict(fit_all_raw, newdata = newdata, type = "logodds")

# log-odds for x3 < -18
pred_log_odds
```

```
##        1
## 1.292977
```

```
# probability for x3 < -18
plogis(pred_log_odds)
```

```
##         1
## 0.7846506
```

$P(X3 < (-18) \mid X1 = 15, X2 = -40)$ on the all standardized data & model:

```
# for x1 = 15, x2 = -40
# outcome x3 < -18

x1_std <- (x1 - mean(x1)) / sd(x1)
x2_std <- (x2 - mean(x2)) / sd(x2)
x3_std <- (x3 - mean(x3)) / sd(x3)

fit_all_std <- Colr(x3_std ~ x1_std + x2_std, order=len_theta)

newdata <- data.frame(x1_std = (15-mean(x1))/sd(x1),
                      x2_std = (-40-mean(x2))/sd(x2),
                      x3_std = (-18-mean(x3))/sd(x3))
pred_log_odds <- predict(fit_all_std, newdata = newdata, type = "logodds")
# log-odds for x3 < -18
pred_log_odds
```

```
##        1
## 1.318627
```

```
# probability for x3 < -18
plogis(pred_log_odds)
```

```
##         1
## 0.7889531
```

We see that the estimated probabilites are almost equal. This means we can use the model fitted on the standardized data and do queries like this.

## Check interpretation of the coefficients:

### Raw outcome - raw predictors

Beta1 (estimated on model using raw data) is the additive change in log-odds for x3 when changing x1 by one unit when holding x2 constant:

```
# fit_all_raw

# beta 1 is the additive change in log odds of x3 when x1 increases by 1 unit, holding x2 constant
coef(fit_all_raw)[1]
```

```
##        x1
## 0.4694353
```

```
# example
# for x1 = 15, x2 = -60
newdata <- data.frame(x1 = 15, x2 = -60)

# predict log odds
pred_log_odds_15 <- predict(fit_all_raw, newdata = newdata, type = "logodds")


# one unit increase in x1 while holding x2 constant
# for x1 = 16, x2 = -60
newdata <- data.frame(x1 = 16, x2 = -60)

# predict log odds
pred_log_odds_16 <- predict(fit_all_raw, newdata = newdata, type = "logodds")


# difference in log odds (for each level of x3! -> proportional)
head(pred_log_odds_16 - pred_log_odds_15)
```

```
##
## x3           [,1]
##   -55.5 0.4694353
##   -54.5 0.4694353
##   -53.5 0.4694353
##   -52.6 0.4694353
##   -51.6 0.4694353
##   -50.6 0.4694353
```

Check on DGP: of course same interpretation

```
# x3 = 1/3 * (x_3_dash - 0.5 * x1 + 1.2 * x2)

# x_3_dash = x3 * 3 + 0.5 * x1 - 1.2 * x2)

# for x1 = 15, x2 = -60
x_3_dash_15 = x3 * 3 + 0.5 * 15 - 1.2 * -60
```

```r
# for x1 = 16, x2 = -60
x_3_dash_16 = x3 * 3 + 0.5 * 16 - 1.2 * -60

# difference in x_3_dash
head(x_3_dash_16 - x_3_dash_15)
```

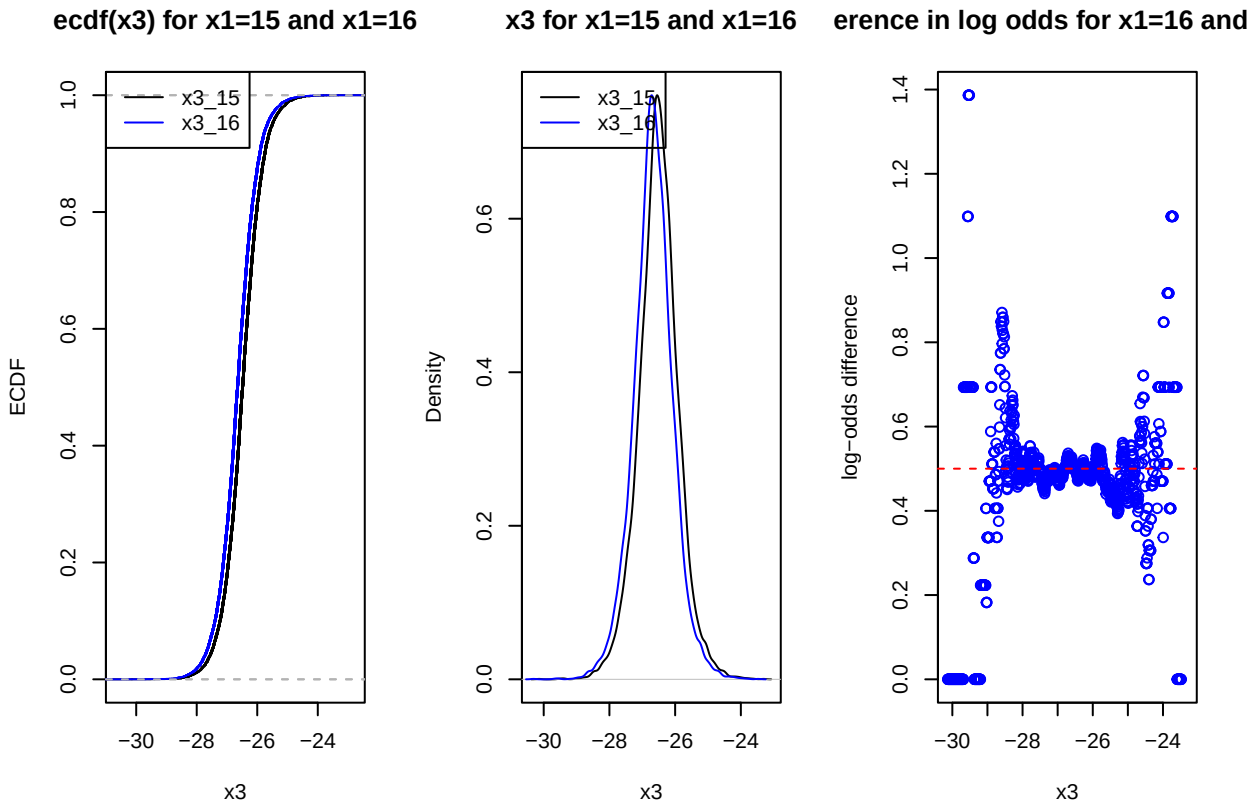
```
## [1] 0.5 0.5 0.5 0.5 0.5 0.5
```

Check on Simulated DGP:

Generate samples from conditional distribution of x3 given x1=15 and x1=16, holding x2 constant:

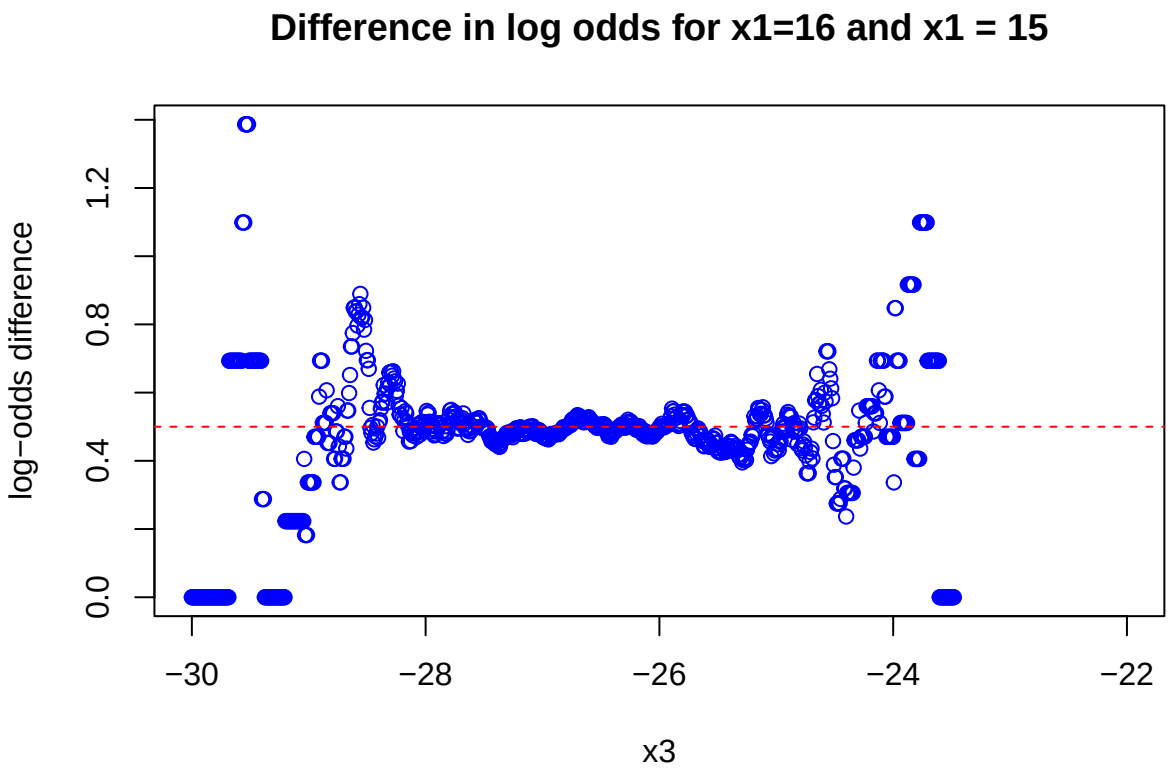
```r
U3 = runif(n)
x_3_dash = qlogis(U3)
#x_3_dash = h_0(x_3) + beta * X_1 + beta * X_2
#x_3_dash = 3 * x_3 + 0.5 * X_1 - 1.2 * X_2

# x3 for x1 = 15, x2 = -60
x3_15 = 1/3 * (x_3_dash - 0.5 * (15) + 1.2 * (-60))

par(mfrow=c(1,3))
#ecdf of x3_15
plot(ecdf(x3_15), main= "ecdf(x3) for x1=15 and x1=16", xlab = "x3", ylab = "ECDF", ylim = c(0, 1))

# x3 for x1 = 16, x2 = -60
x3_16 = 1/3 * (x_3_dash - 0.5 * (16) + 1.2 * (-60))
lines(ecdf(x3_16), col = 'blue')
legend("topleft", legend = c("x3_15", "x3_16"), col = c("black", "blue"), lty = 1)

plot(density(x3_15), main = "x3 for x1=15 and x1=16", xlab = "x3", ylab = "Density")
lines(density(x3_16), col = "blue")
legend("topleft", legend = c("x3_15", "x3_16"), col = c("black", "blue"), lty = 1)

# check log odds for x3_16 at x1 = 15, x2 = -60, x3_16 = -26
p <- length(x3_16[x3_16 < -26])/n
log_odds_16 <- log(p/(1-p))

# check log odds for x3_15 at x1 = 15, x2 = -60, x3_15 = -26
p <- length(x3_15[x3_15 < -26])/n
log_odds_15 <- log(p/(1-p))

# check diff in log odds:
# log_odds_16 - log_odds_15


# check log odds for 1000 values of x3

outcome <- seq(min(x3_15), max(x3_15), length.out = 1000)
lodds <- numeric(length(outcome))
for(i in 1:length(outcome)){
  o <- outcome[i]
  p_15 <- length(x3_15[x3_15 < o])/n
  p_16 <- length(x3_16[x3_16 < o])/n
```

```r
  log_odds_15 <- log(p_15/(1-p_15))
  log_odds_16 <- log(p_16/(1-p_16))
  lodds[i] <- log_odds_16 - log_odds_15
}
# lodds

plot(outcome, lodds, main = "Difference in log odds for x1=16 and x1 = 15", xlab = "x3", ylab = "log-odd
abline(h=0.5, col = "red", lty = 2)
```




Oder direkt mit ECDF (geht schneller):

```r
outcome <- seq(-30, -22, length.out = 1000)


# log-odds for x3_15
lo15 <- log(ecdf(x3_15)(outcome)/(1-ecdf(x3_15)(outcome)))

# log-odds for x3_16
lo16 <- log(ecdf(x3_16)(outcome)/(1-ecdf(x3_16)(outcome)))

# difference in log odds
# lo16 - lo15

plot(outcome, lo16 - lo15, main = "Difference in log odds for x1=16 and x1 = 15", xlab = "x3", ylab = "
abline(h=0.5, col = "red", lty = 2)
```




**Raw outcome - standardized predictors**

predict logodds for 1 sd(x1) increase in x1 when holding x2 constant

```
# estimated coef for x1 standardized
coef(fit_x3_raw)[1]
```

```
##   x1_std
## 1.975725
```

```
# for x1 = 20 (standardized), x2 = -0.5
newdata1 <- data.frame(x1_std = (20 - mean(x1))/sd(x1), x2_std = -0.5)
# predict log odds
pred_log_odds_1 <- predict(fit_x3_raw, newdata = newdata1, type = "logodds")

# one sd(x1) increase in x1 while holding x2 constant
newdata2 <- data.frame(x1_std = (20 + sd(x1) - mean(x1))/sd(x1), x2_std = -0.5)

# predict log odds
pred_log_odds_2 <- predict(fit_x3_raw, newdata = newdata2, type = "logodds")

# difference in log odds (for each level of x3! -> proportional)
head(pred_log_odds_2 - pred_log_odds_1)
```

```
##
## x3          [,1]
##   -55.5 1.975725
##   -54.5 1.975725
##   -53.5 1.975725
##   -52.6 1.975725
##   -51.6 1.975725
##   -50.6 1.975725
```

**Standardized outcome - standardized predictors**

predict logodds for standardized x3 for 1 sd(x1) increase in x1 when holding x2 constant

```
# estimated coef for x1 standardized
coef(fit_all_std)[1]
```

```
##   x1_std
## 1.976529
```

```
# for x1 = 20 (standardized), x2 = -0.5
newdata1 <- data.frame(x1_std = (20 - mean(x1))/sd(x1), x2_std = -0.5)
# predict log odds
pred_log_odds_1 <- predict(fit_all_std, newdata = newdata1, type = "logodds")
# one sd(x1) increase in x1 while holding x2 constant
newdata2 <- data.frame(x1_std = (20 + sd(x1) - mean(x1))/sd(x1), x2_std = -0.5)
# predict log odds
pred_log_odds_2 <- predict(fit_all_std, newdata = newdata2, type = "logodds")
# difference in log odds (for each level of x3! -> proportional)
head(pred_log_odds_2 - pred_log_odds_1)
```

```
##
## x3_std      [,1]
##   -2.21 1.976529
##   -2.09 1.976529
##   -1.98 1.976529
##   -1.87 1.976529
##   -1.75 1.976529
##   -1.64 1.976529
```

Check on DGP: of course same interpretation. In reality the transformation function h(x3_std) would be different than h(x3) because the x-axis is scaled.

```
# check on DGP: of course same interpretation
# x3 = 1/3 * (x_3_dash - 0.5 * x1 + 1.2 * x2)
# x_3_dash = x3 * 3 + 0.5 * X_1 - 1.2 * X_2)
# for x1 = 20, x2 = -60
x_3_dash_15 = x3_std * 3 + 0.5 * (20) - 1.2 * (-60)
# for x1 = 20+sd(x1), x2 = -60
x_3_dash_16 = x3_std * 3 + 0.5 * (20 + sd(x1)) - 1.2 * (-60)
# difference in x_3_dash
head(x_3_dash_16 - x_3_dash_15)
```

```
## [1] 2.069841 2.069841 2.069841 2.069841 2.069841 2.069841
```

```
# the difference in log odds is the same the dgp coefficient times the standard deviation of the raw x1
0.5 * sd(x1)
```

```
## [1] 2.069841
```

### Scaling simulation with non linear transformation function for x3

In the previous simulations I always used a linear transformation function for x3. Now I will use a non-linear transformation function for x3 and check if the conclusion is still the same. (yes it is still the same)

```
n <- 1000

X_1_A = rnorm(n, 3, 2.5)
X_1_B = rnorm(n, 7, 1)
x1 = ifelse(sample(1:2, replace = TRUE, size = n) == 1, X_1_A, X_1_B)


U2 = runif(n)
x_2_dash = qlogis(U2)

#x_2_dash = h_0(x_2) + beta * X_1
#x_2_dash = 0.42 * x_2 + 2 * X_1

x2 = 1/0.42 * (x_2_dash + 0.4 * x1)
```

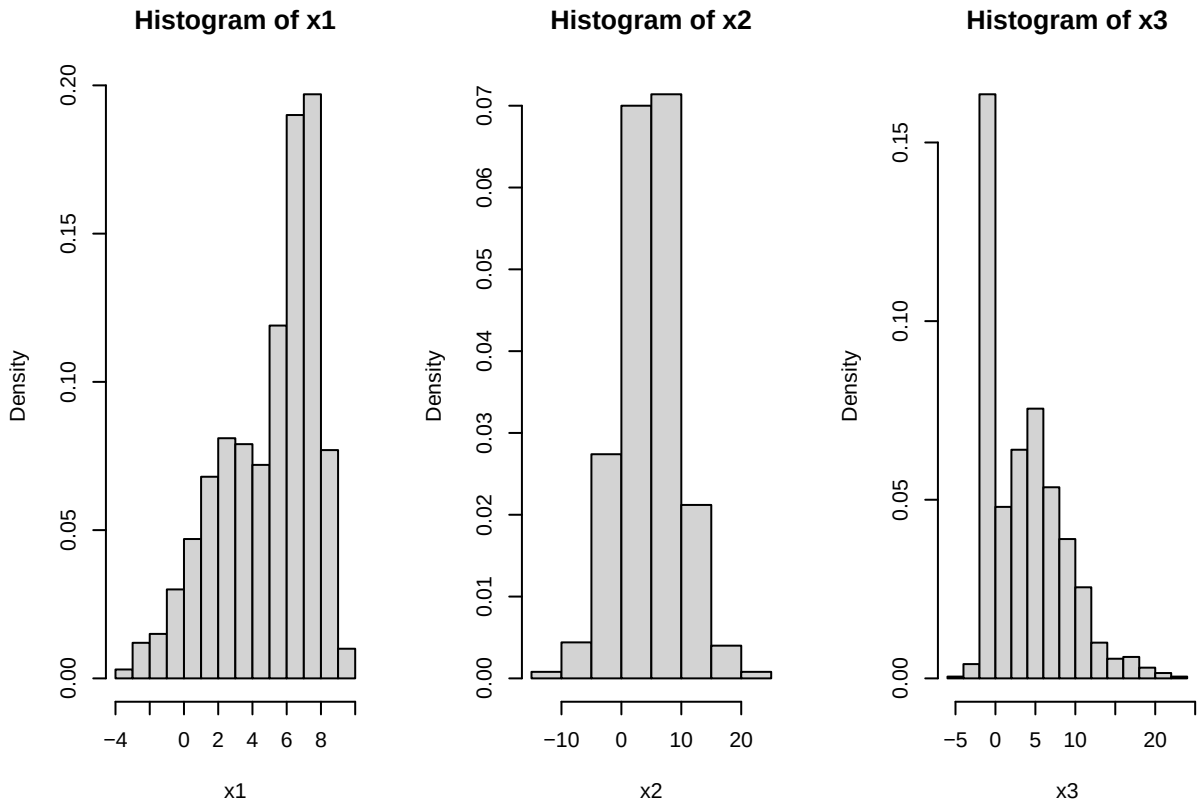
Non-linear Softplus trafo h=log(1+exp(x3)):

```
U3 = runif(n)
x_3_dash = qlogis(U3)

#x_3_dash = h_0(x_3) + beta * X_2
#x_3_dash = 3 * x_2 + 0.5 * X_1 - 1.2 * X_2


x3 = log(exp(x_3_dash - 0.5 * x1 + 1.2 * x2)-1)
```

```
## Warning in log(exp(x_3_dash - 0.5 * x1 + 1.2 * x2) - 1): NaNs wurden erzeugt
```

```
# replace the NaN with 0
x3[is.na(x3)] <- 0
```

```
par(mfrow=c(1,3))

hist(x1, freq = FALSE)
hist(x2, freq = FALSE)
hist(x3, freq = FALSE)
```




summary table of the 3 variables raw and standardized:

```
x1_std <- (x1 - mean(x1)) / sd(x1)
x2_std <- (x2 - mean(x2)) / sd(x2)
x3_std <- (x3 - mean(x3)) / sd(x3)

# summary table

summary_table <- data.frame(
  Variable = c("x1", "x2", "x3"),
  Mean = c(mean(x1), mean(x2), mean(x3)),
  SD = c(sd(x1), sd(x2), sd(x3)),
  Mean_std = c(mean(x1_std), mean(x2_std), mean(x3_std)),
  SD_std = c(sd(x1_std), sd(x2_std), sd(x3_std))
)

# round numeric values
summary_table[, 2:5] <- round(summary_table[, 2:5], 2)

kable(summary_table, caption = "Summary table of the 3 variables raw and standardized")
```

Table 2: Summary table of the 3 variables raw and standardized

| Variable | Mean | SD | Mean_std | SD_std |
|---|---|---|---|---|
| x1 | 4.97 | 2.76 | 0 | 1 |
| x2 | 4.72 | 4.99 | 0 | 1 |
| x3 | 3.98 | 4.46 | 0 | 1 |

Analyze Colr(x3 ~ x1 + x2, order=len_theta) :

```
fit_all_raw <- Colr(x3 ~ x1 + x2, order=len_theta)
fit_all_std <- Colr(x3_std ~ x1_std + x2_std, order=len_theta)
fit_x3_raw <- Colr(x3 ~ x1_std + x2_std, order=len_theta)
fit_x3_std <- Colr(x3_std ~ x1 + x2, order=len_theta)
```

```
coef(fit_all_raw)
```

```
##         x1         x2
##  0.2963810 -0.7331257
```

```
coef(fit_all_std)
```

```
##     x1_std     x2_std
##  0.8172404 -3.6665582
```

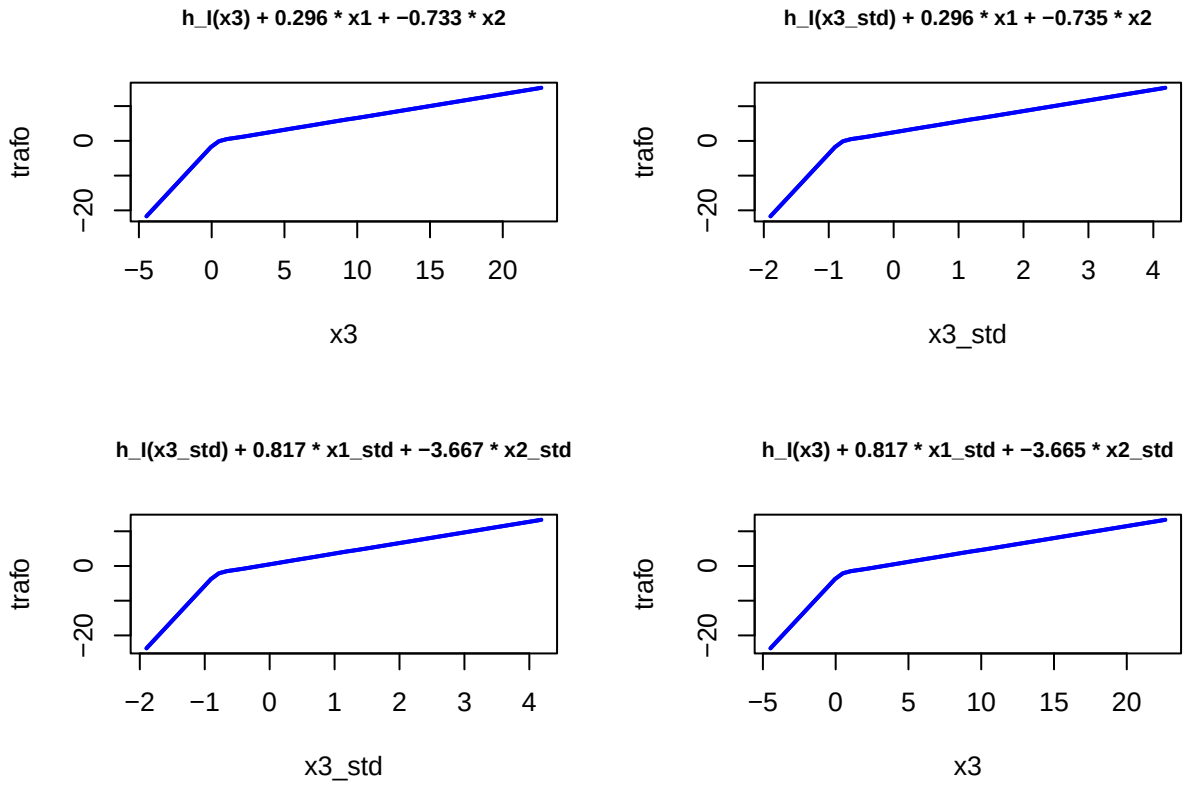
```
coef(fit_x3_raw)
```

```
##     x1_std     x2_std
##  0.8168801 -3.6646780
```

```
coef(fit_x3_std)
```

```
##         x1         x2
##  0.2962607 -0.7350899
```

```
# plot baseline trafo
par(mfrow=c(2,2))
temp = model.frame(fit_all_raw)[1:2,-1, drop=FALSE]
plot(fit_all_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3) + ", round(coef(fit_all_raw)[1], 3), " * x1 + ", round(coef(fit_all_raw)[2], 3
temp = model.frame(fit_x3_std)[1:2,-1, drop=FALSE]
plot(fit_x3_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3_std) + ", round(coef(fit_x3_std)[1], 3), " * x1 + ", round(coef(fit_x3_std)[2]

temp = model.frame(fit_all_std)[1:2,-1, drop=FALSE]
plot(fit_all_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3_std) + ", round(coef(fit_all_std)[1], 3), " * x1_std + ", round(coef(fit_all_s

temp = model.frame(fit_x3_raw)[1:2,-1, drop=FALSE]
plot(fit_x3_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x3) + ", round(coef(fit_x3_raw)[1], 3), " * x1_std + ", round(coef(fit_x3_raw)[2]
```




We can see that the results are the same as with a linear transformation function. The coefficients are still the same regardless if we use the raw outcome or the standardized outcome x3.

## Simulate and scale for Linear regression

```
len_theta <- 20

# Simulate and scale for Linear Regression

x1 <- runif(1000, 0, 50)
y <- 2*x1 + rnorm(1000, 0, 5)

par(mfrow=c(1,2))
plot(x1, y)

lm1 <- lm(y ~x1)
lines(x1, predict(lm1), col = "red")

## scale x1 and y

x1s <- scale(x1)
summary(x1s)
```

```
##        V1
```

```
##  Min.   :-1.76663
##  1st Qu.:-0.88011
##  Median : 0.04262
##  Mean   : 0.00000
##  3rd Qu.: 0.84706
##  Max.   : 1.71960
```

```
sd(x1s)
```

```
## [1] 1
```

```
ys <- scale(y)

lm2 <- lm(ys ~ x1s)
summary(lm1)
```

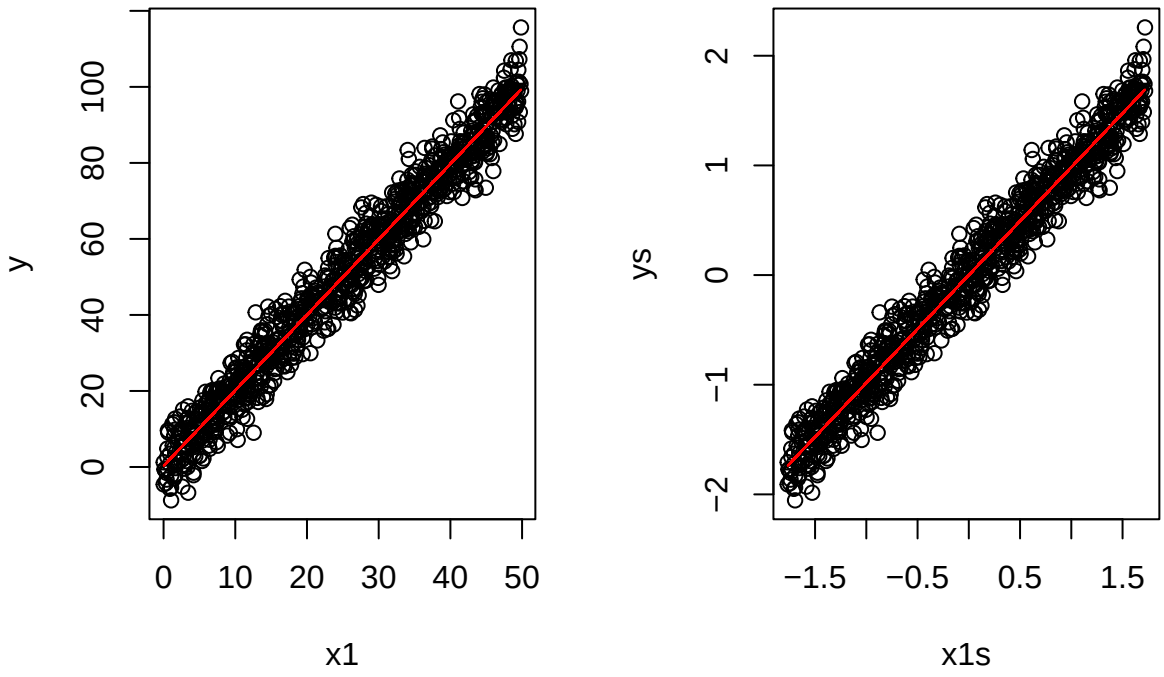
```
##
## Call:
## lm(formula = y ~ x1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.2398  -3.4274  -0.0156   3.5157  16.3636
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.3147     0.3339   0.943    0.346
## x1            1.9852     0.0115 172.619   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.196 on 998 degrees of freedom
## Multiple R-squared:  0.9676, Adjusted R-squared:  0.9676
## F-statistic: 2.98e+04 on 1 and 998 DF,  p-value: < 2.2e-16
```

```
summary(lm2)
```

```
##
## Call:
## lm(formula = ys ~ x1s)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56290 -0.11880 -0.00054  0.12186  0.56719
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.333e-16  5.696e-03     0.0        1
## x1s         9.837e-01  5.698e-03   172.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.1801 on 998 degrees of freedom
## Multiple R-squared:  0.9676, Adjusted R-squared:  0.9676
## F-statistic: 2.98e+04 on 1 and 998 DF,  p-value: < 2.2e-16
```

```r
plot(x1s, ys)
lines(x1s, predict(lm2), col = "red")
```

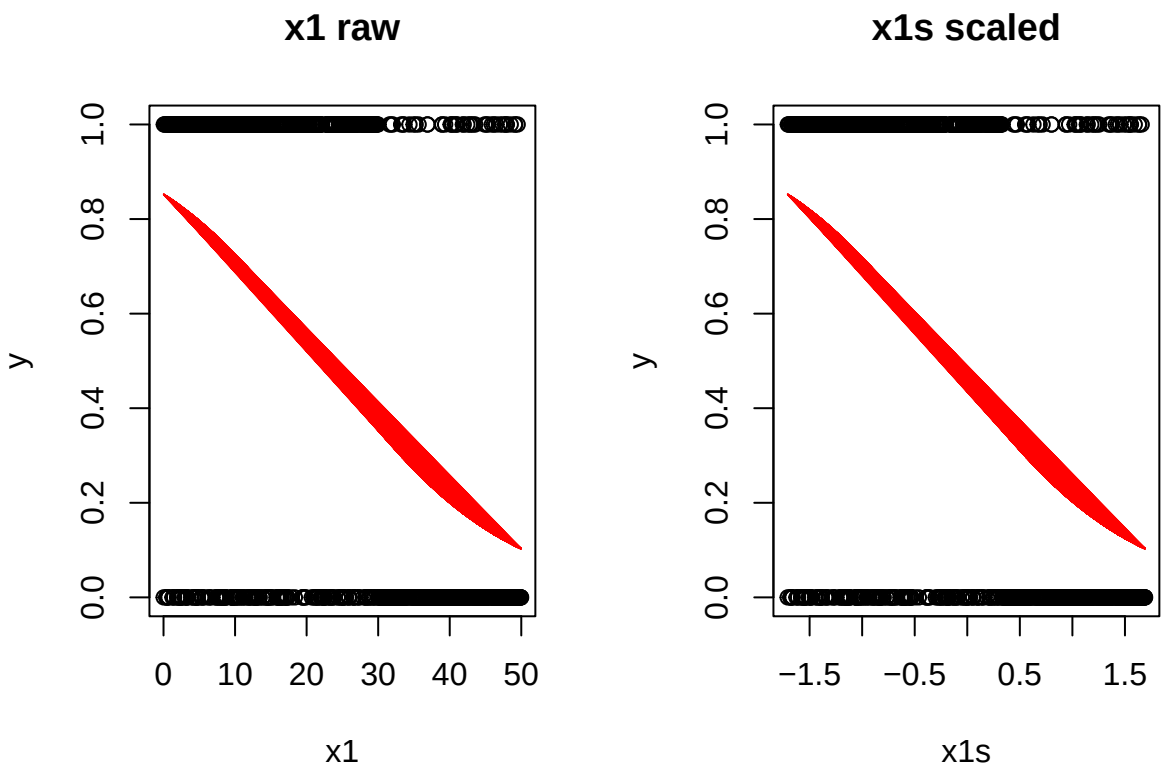



Coefficient for the scaled variables is different.

### Simulate and scale for Logistic regression

```r
x1 <- runif(1000, 0, 50)
y <- sapply(x1, function(x) ifelse(x < 30, rbinom(1, 1, prob=0.7), rbinom(1, 1, prob=0.1)) )

par(mfrow=c(1,2))
plot(x1, y, main= "x1 raw")


glm1 <- glm(y ~x1, family = binomial)
# summary(glm1)

# add glm1 to plot above
lines(x1, predict(glm1, type = "response"), col = "red")
```

```r
# scale x1

x1s <- scale(x1)
# summary(x1s)

glm2 <- glm(y ~ x1s, family = binomial)
# summary(glm2)

plot(x1s, y, main ="x1s scaled")
lines(x1s, predict(glm2, type = "response"), col = "red")
```




```r
# raw covariate
summary(x1)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.03162 12.21944 25.07448 25.12411 37.89299 49.96926
```

```r
# scaled covariate
summary(x1s)
```

```
##        V1
##  Min.   :-1.707977
##  1st Qu.:-0.878385
```

```
##  Median :-0.003378
##  Mean   : 0.000000
##  3rd Qu.: 0.869143
##  Max.   : 1.691142
```

In Logistic Regression, scaling the Covariate changes the coefficients.

```
# coefficients differ!
coef(glm1)
```

```
## (Intercept)          x1
##  1.75492539 -0.07847395
```

```
coef(glm2)
```

```
## (Intercept)         x1s
##  -0.2166624  -1.1528883
```

```
# the coefficient for the scaled data is approximately the coefficient for
# the raw data multiplied by the standard deviation of the raw data
coef(glm1)[2]*sd(x1)
```

```
##        x1
## -1.152888
```

```
# the intercept is the old intercept + the old mean multiplied by the old
# coefficient
coef(glm1)[1] + coef(glm1)[2] * mean(x1)
```

```
## (Intercept)
##  -0.2166624
```

Let $X$ be a continuous predictor and $Y \in \{0, 1\}$ a binary outcome.

We model the relationship between them using a logistic regression:

$$\text{logit}(\mathbb{P}(Y = 1 \mid X)) = \beta_0 + \beta_1 X$$

Now suppose we standardize $X$ to obtain $X_{\text{std}} = \frac{X - \mu_X}{\sigma_X}$, where $\mu_X$ and $\sigma_X$ are the sample mean and standard deviation of $X$, respectively.

Then the logistic model becomes:

$$\text{logit}(\mathbb{P}(Y = 1 \mid X_{\text{std}})) = \beta_0^* + \beta_1^* X_{\text{std}}$$

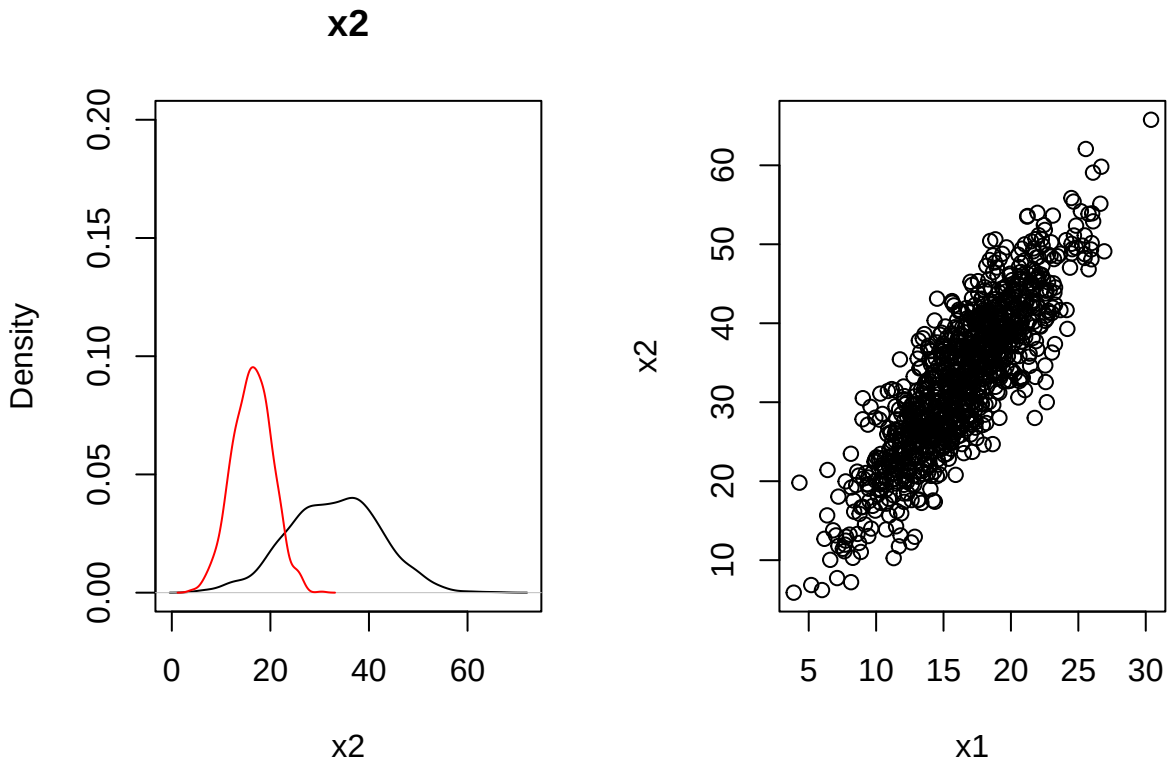
The coefficients relate via:

$$\beta_1^* = \beta_1 \cdot \sigma_X$$

This means that $\beta_1^*$ represents the change in log-odds of $Y = 1$ for a one **standard deviation** increase in $X$, whereas $\beta_1$ represents the change in log-odds per one **unit** increase in raw $X$.

**Simulate for Colr**

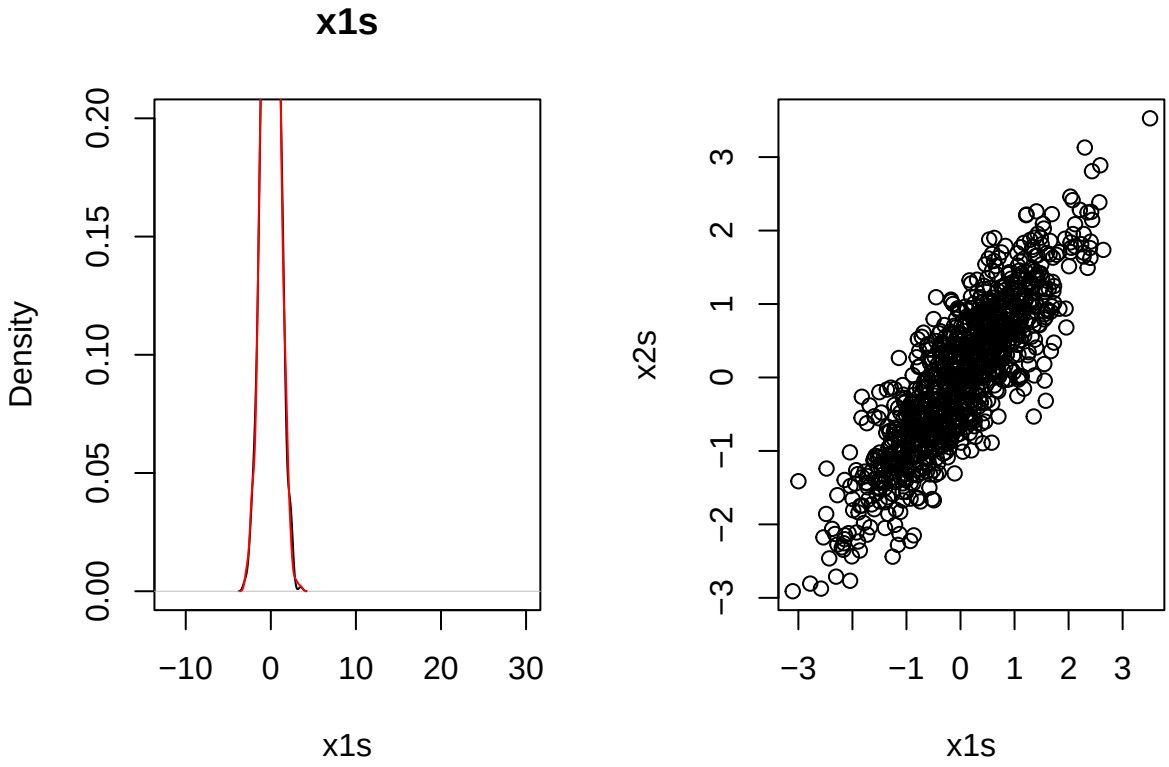
```
x1 <- rnorm(1000, 16, 4)
x2 <- x1 * (2) + rnorm(1000, 0, 5)

# summary(x1)
# summary(x2)

par(mfrow = c(1,2))

# plot densities
plot(density(x2), main = "x2", xlab = "x2", ylab = "Density",  ylim = c(0, 0.2))
lines(density(x1), col = "red")

plot(x1, x2)
```




Standardize samples:

```
x1s <- (x1 - mean(x1)) / sd(x1)
x2s <- (x2 - mean(x2)) / sd(x2)

# summary(x1s)
# summary(x2s)

par(mfrow=c(1,2))
```

```r
# plot densities
plot(density(x1s), main = "x1s", xlab = "x1s", ylab = "Density", xlim = c(-12, 30), ylim = c(0, 0.2))
lines(density(x2s), col = "red")

plot(x1s, x2s)
```




```r
fit_raw <- Colr(x2 ~ x1, order=len_theta)
coef(fit_raw)
```

```
##         x1
## -0.7338346
```

```r
fit_x2_std <- Colr(x2s ~ x1, order=len_theta)
coef(fit_x2_std)
```

```
##         x1
## -0.7338346
```

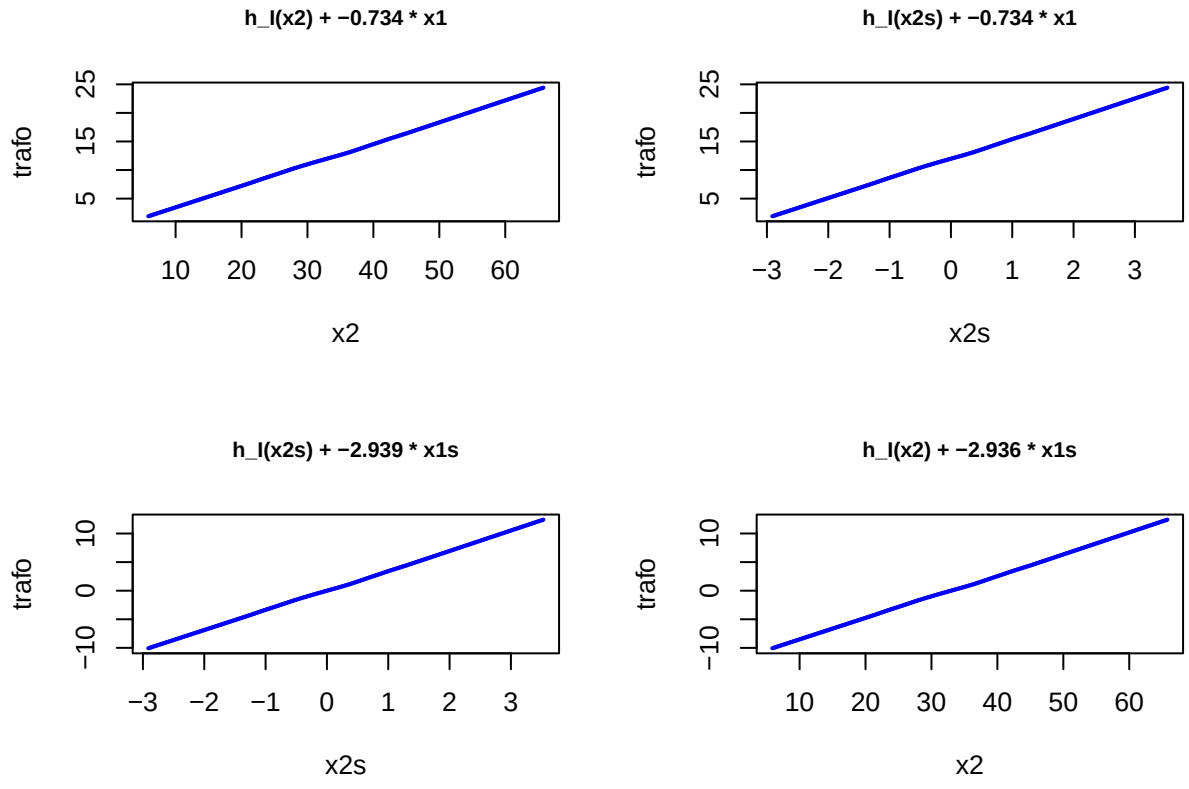
```r
fit_std <- Colr(x2s ~ x1s, order=len_theta)
coef(fit_std)
```

```
##       x1s
## -2.938722
```

```
fit_x2_raw <- Colr(x2 ~ x1s, order=len_theta)
coef(fit_x2_raw)
```

```
##       x1s
## -2.935614
```

```
# plot baseline trafo

par(mfrow=c(2,2))
temp = model.frame(fit_raw)[1:2,-1, drop=FALSE]
plot(fit_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2) + ", round(coef(fit_raw)[1], 3), " * x1"), cex.main=0.8)
temp = model.frame(fit_x2_std)[1:2,-1, drop=FALSE]
plot(fit_x2_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2s) + ", round(coef(fit_x2_std)[1], 3), " * x1"), cex.main=0.8)
temp = model.frame(fit_std)[1:2,-1, drop=FALSE]
plot(fit_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2s) + ", round(coef(fit_std)[1], 3), " * x1s"), cex.main=0.8)
temp = model.frame(fit_x2_raw)[1:2,-1, drop=FALSE]
plot(fit_x2_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(x2) + ", round(coef(fit_x2_raw)[1], 3), " * x1s"), cex.main=0.8)
```



# Real Data

## Data Preparation

Read raw data:

```
library(ncdf4)

# read the file "enso_son.nc"  without raster stored in the same directory
enso <- nc_open("../data/enso_son.nc")
enso_y <- ncvar_get(enso, "Year")
enso_raw <- ncvar_get(enso, "enso")

# print the timeseries of enso
# plot(enso_y, enso_v, type = "l", xlab = "Year", ylab = "ENSO")


# read the file "iod_son.nc" stored in the same directory

iod <- nc_open("../data/iod_son.nc")
iod_y <- ncvar_get(iod, "Year")
iod_raw <- ncvar_get(iod, "iod")

# print the timeseries of iod
# plot(iod_y, iod_v, type = "l", xlab = "Year", ylab = "IOD")


# read the file "precip_au_son.nc" stored in the same directory

au <- nc_open("../data/precip_au_son.nc")
au_y <- ncvar_get(au, "year")
au_raw <- ncvar_get(au, "precip")
# hist(au_raw)
```

Standardize data:

```
# standardize (zero to mean, unit variance)
ENSO_std <- (enso_raw - mean(enso_raw))/sd(enso_raw)
IOD_std <- (iod_raw - mean(iod_raw))/sd(iod_raw)
AU_std <- (au_raw - mean(au_raw))/sd(au_raw)
```

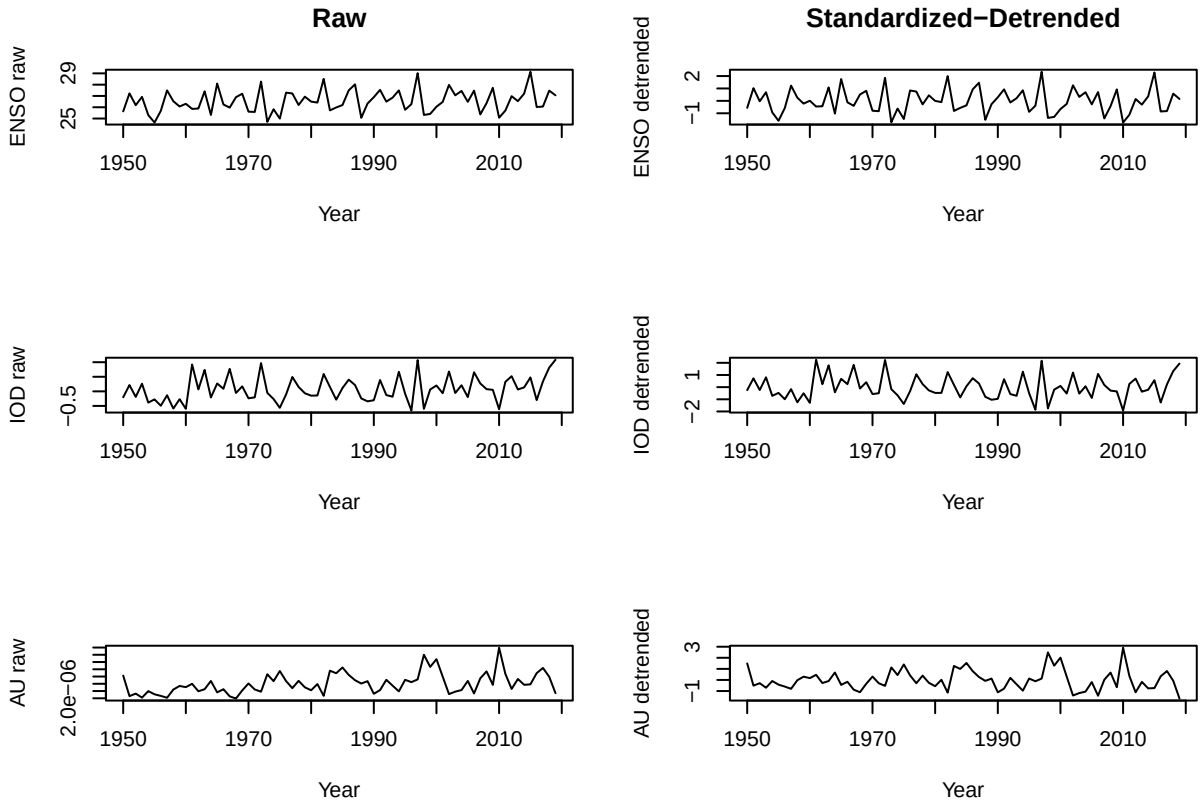
Detrend standardized data:

```
ENSO_detrended <- residuals(lm(ENSO_std ~ enso_y))
IOD_detrended <- residuals(lm(IOD_std ~ iod_y))
AU_detrended <- residuals(lm(AU_std ~ au_y))
```

Plot timeseries:

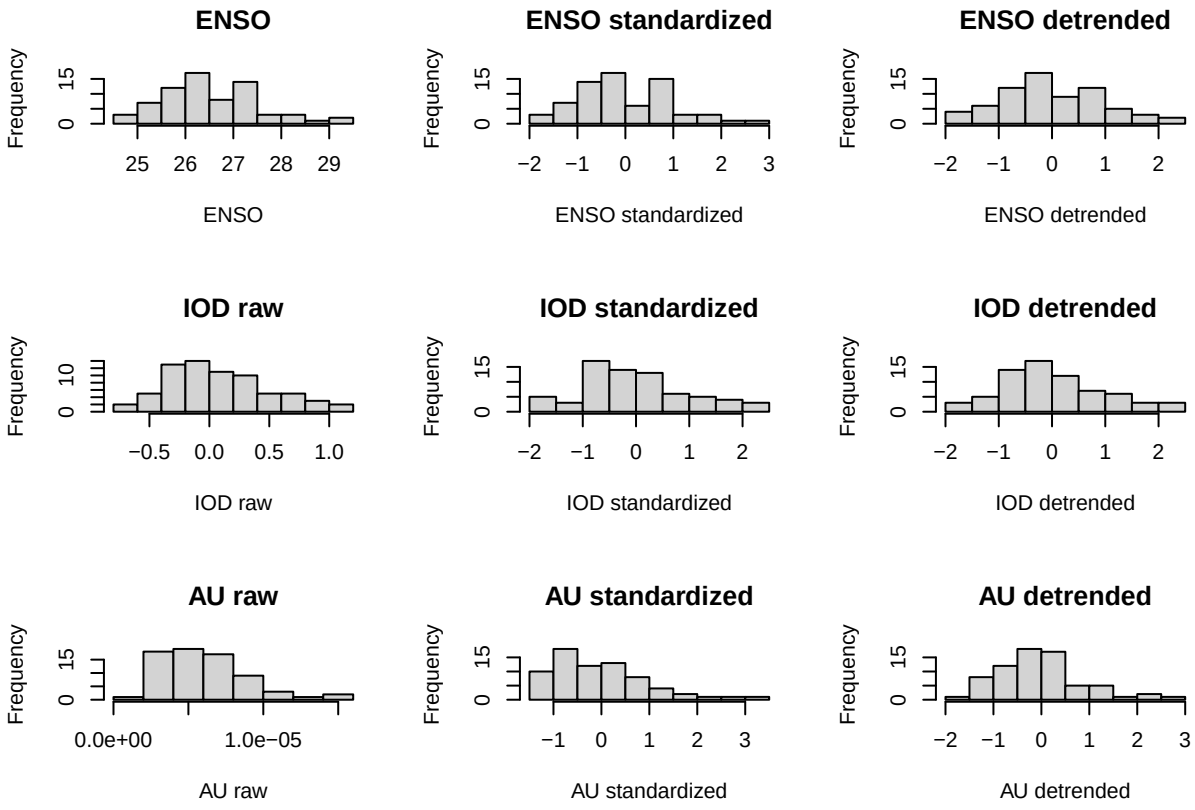
```
# plot raw timeseries
par(mfrow=c(3,2))
plot(enso_y, enso_raw, type = "l", xlab = "Year", ylab = "ENSO raw",
```

```
     main = "Raw")
plot(enso_y, ENSO_detrended, type = "l", xlab = "Year", ylab = "ENSO detrended",
     main = "Standardized-Detrended")

plot(iod_y, iod_raw, type = "l", xlab = "Year", ylab = "IOD raw")
plot(iod_y, IOD_detrended, type = "l", xlab = "Year", ylab = "IOD detrended")

plot(au_y, au_raw, type = "l", xlab = "Year", ylab = "AU raw")
plot(au_y, AU_detrended, type = "l", xlab = "Year", ylab = "AU detrended")
```




Plot histograms:

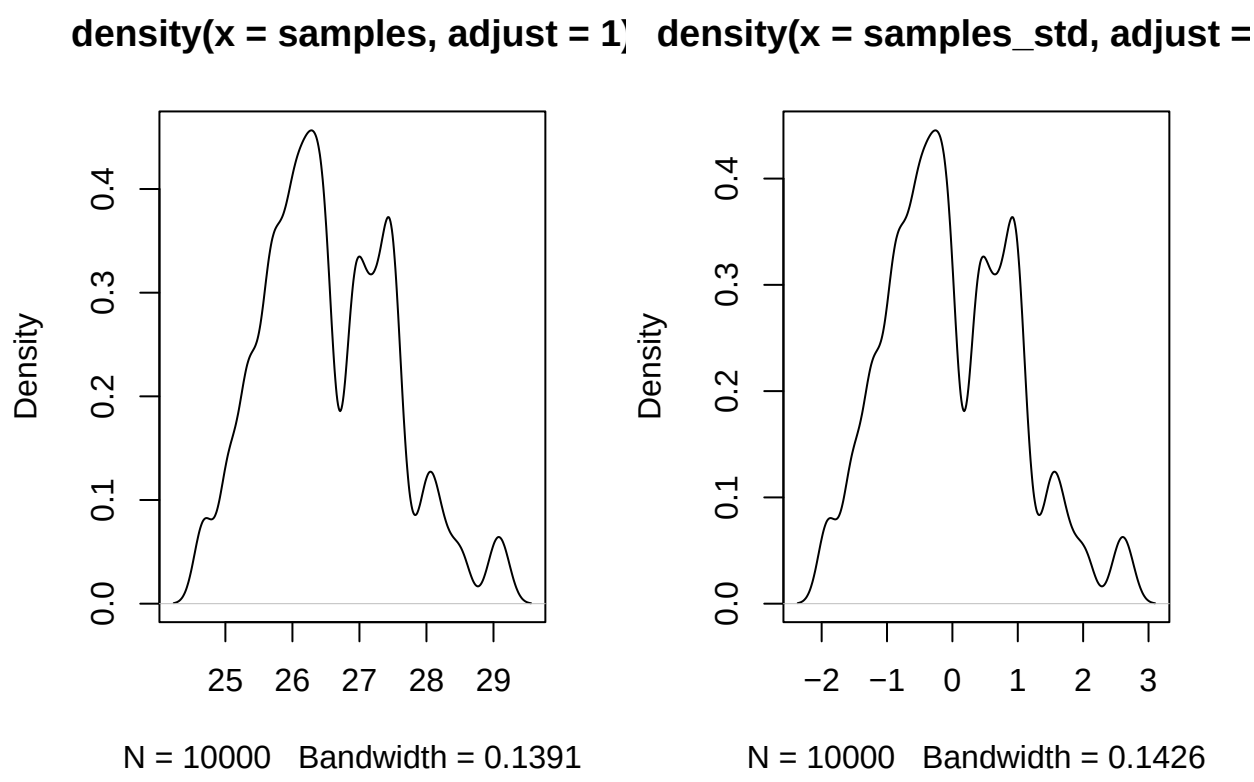
```
# 3 histograms of the raw data
par(mfrow=c(3,3))
hist(enso_raw, main = "ENSO", xlab = "ENSO")
hist(ENSO_std, main = "ENSO standardized", xlab = "ENSO standardized")
hist(ENSO_detrended, main = "ENSO detrended", xlab = "ENSO detrended")

hist(iod_raw, main = "IOD raw", xlab = "IOD raw")
hist(IOD_std, main = "IOD standardized", xlab = "IOD standardized")
hist(IOD_detrended, main = "IOD detrended", xlab = "IOD detrended")

hist(au_raw, main = "AU raw", xlab = "AU raw")
hist(AU_std, main = "AU standardized", xlab = "AU standardized")
hist(AU_detrended, main = "AU detrended", xlab = "AU detrended")
```




## Impact of Scaling (real data)

Scaling only changes the x-Axis values, shape and value of density remains unchanged.

```
# Set seed for reproducibility
set.seed(123)

samples <- sample(enso_raw, replace = TRUE, size = 10000)
samples_std <- (samples- mean(samples))/sd(samples)

par(mfrow=c(1,2))
plot(density(samples, adjust = 1))
plot(density(samples_std, adjust = 1))
```




We want to model the impact of ENSO and IOD on AU. The model is given by:

$\mathrm{F}(AU \mid ENSO, IOD) = F_z(h(AU) + \beta_1 ENSO + \beta_2 IOD)$

These are the scales of the raw variables:

```
# ENSO
sd(enso_raw)
```

```
## [1] 0.9968555
```

```
# IOD
sd(iod_raw)
```

```
## [1] 0.4280147
```

```
# AU
sd(au_raw)
```

```
## [1] 2.855879e-06
```

Standard deviation of ENSO is almost 1, therefore scaling might not have a big effect (only a shift). IOD is 0.42 and AU is very small.

Fit a Colr model with the raw data:

```
len_theta <- 20

## only raw data
fit_raw = Colr(au_raw ~ enso_raw + iod_raw, order=len_theta)
coef(fit_raw)
```

```
##   enso_raw    iod_raw
## 0.68235266 0.06322657
```

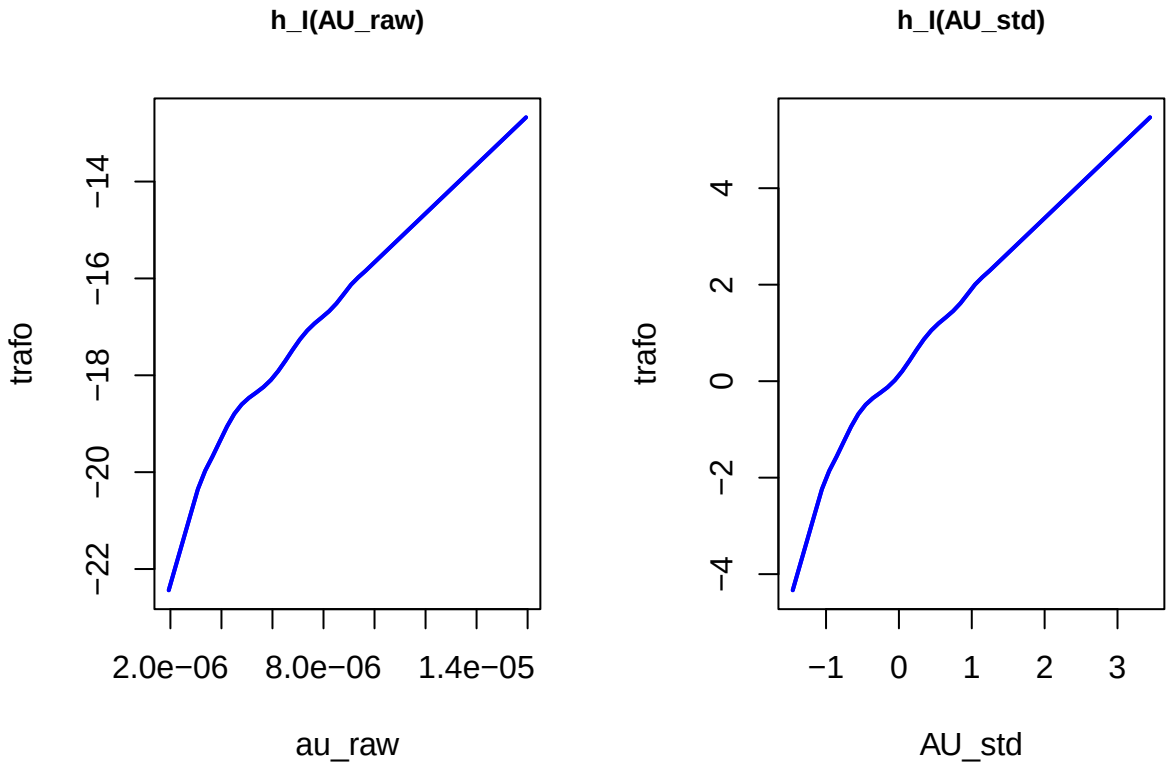
Fit a Colr model with the standardized data:

```
## only standardized data
fit_std = Colr(AU_std ~ ENSO_std + IOD_std, order=len_theta)
coef(fit_std)
```

```
##     ENSO_std      IOD_std
##  0.749442651 -0.007374579
```

Comparison of baseline Transformation functions:

```
par(mfrow=c(1,2))
temp = model.frame(fit_raw)[1:2,-1, drop=FALSE]
plot(fit_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main='h_I(AU_raw)', cex.main=0.8)


temp = model.frame(fit_std)[1:2,-1, drop=FALSE]
plot(fit_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main='h_I(AU_std)', cex.main=0.8)
```




In both plots, the x-axis scale changes because the response scale is changed. And also the y-axis (latent scale) is changed, because the scale of the predictors is changed. For example, the raw ENSO is on average around value 26, and when standardized around 0, this changes the y-axis intercept. The shape of the transformation function is the same.

### Mixed scales

***All raw model:***

$\mathrm{F}(AU_{raw} \mid ENSO_{raw}, IOD_{raw}) = F_z(h(AUraw) + \beta_1 ENSO_{raw} + \beta_2 IOD_{raw})$

```
fit_all_raw = Colr(au_raw ~ enso_raw + iod_raw, order=len_theta)
round(coef(fit_all_raw), 4)
```

```
## enso_raw  iod_raw
##   0.6824   0.0632
```

Interpretation: $\beta_1$ is the additive change in log-odds for AU for a 1 unit change in ENSO, when holding IOD constant. The reference level for ENSO is 0.

Example: for ENSO=24, the log-odds of AU is $h(AU) + \beta_1 * 24 + \beta_2 * IOD$.

***Standardized Outcome, raw predictors:***

$\mathrm{F}(AU_{std} \mid ENSO_{raw}, IOD_{raw}) = F_z(h(AUstd) + \beta_1 ENSO_{raw} + \beta_2 IOD_{raw})$

```
fit_AU_std = Colr(AU_std ~ enso_raw + iod_raw, order=len_theta)
round(coef(fit_AU_std), 4)
```

```
## enso_raw  iod_raw
##   0.7481  -0.0097
```

***All standardized model:***

$$\mathrm{F}(AU_{\text{std}} \mid ENSO_{\text{std}}, IOD_{\text{std}}) = F_z\left(h(AU_{\text{std}}) + \beta_1 * \frac{ENSO - \text{mean}(ENSO)}{\text{sd}(ENSO)} + \beta_2 * \frac{IOD - \text{mean}(IOD)}{\text{sd}(IOD)}\right)$$

```
## [1] "Mean ENSO: 26.55   Sd ENSO: 1"
```

```
## [1] "Mean IOD: 0.08   Sd IOD: 0.43"
```

```
fit_all_std = Colr(AU_std ~ ENSO_std + IOD_std, order=len_theta)
round(coef(fit_all_std), 4)
```

```
## ENSO_std  IOD_std
##   0.7494  -0.0074
```

Interpretation: $\beta_1$ is the additive change in log-odds for AU for a 1 unit change in standardized ENSO (in other words, for a 1 standard deviation change of ENSO), when holding IOD constant. The reference level for ENSO is 26.55 (mean of ENSO).

Example: for ENSO=26.55, the log-odds of AU is $h(AU) + \beta_1 * (26.55 - 26.55)/1 + \beta_2 * IOD = h(AU) + 0 + \beta_2 * IOD$.

***AU raw, predictors standardized:***

$$\mathrm{F}(AU_{\text{raw}} \mid ENSO_{\text{std}}, IOD_{\text{std}}) = F_z\left(h(AU_{\text{raw}}) + \beta_1 * \frac{ENSO - \text{mean}(ENSO)}{\text{sd}(ENSO)} + \beta_2 * \frac{IOD - \text{mean}(IOD)}{\text{sd}(IOD)}\right)$$

```
fit_AU_raw = Colr(au_raw ~ ENSO_std + IOD_std, order=len_theta)
round(coef(fit_AU_raw), 4)
```

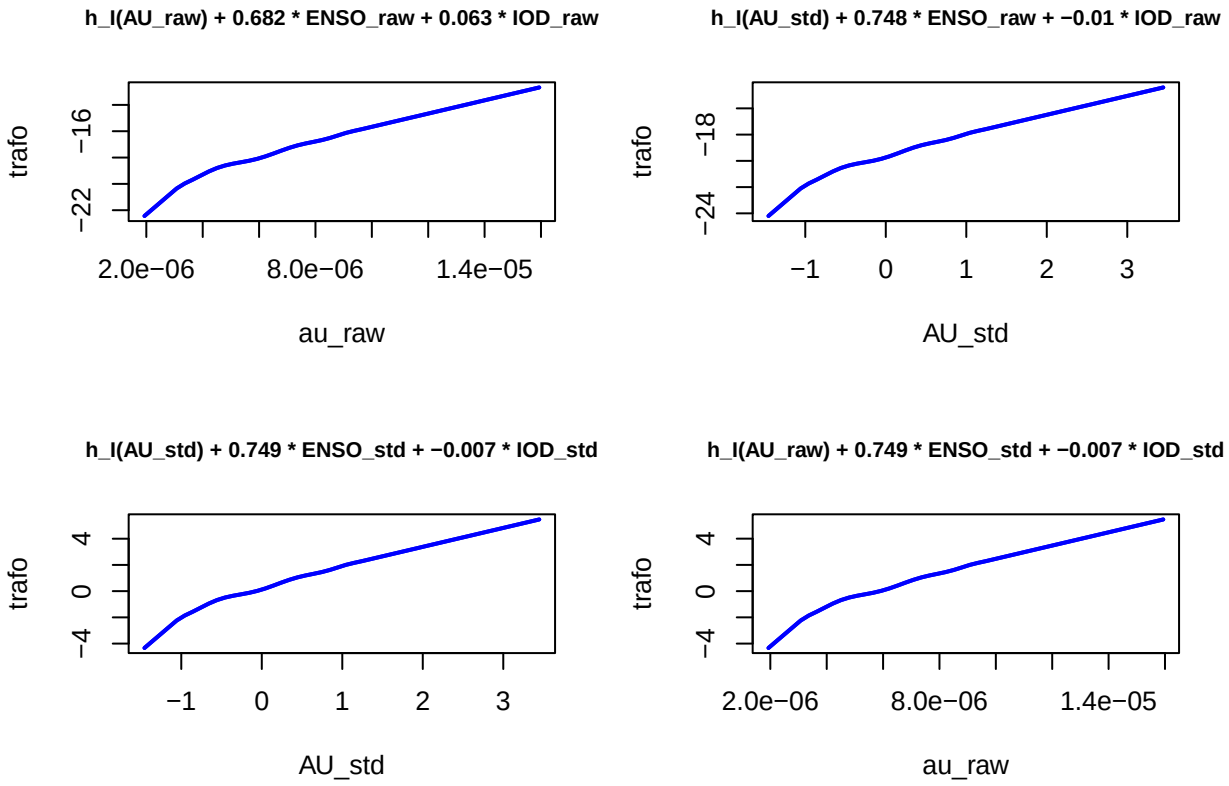
```
## ENSO_std  IOD_std
##   0.7491  -0.0075
```

The coefficients $\beta_1$ and $\beta_2$ are the same as in the previous model with the standardized outcome. This means, that when the predictors are standardized, it does not matter, if the outcome is standardized or not.

Interpretation: same as in the all standardized model above.

```
par(mfrow = c(2,2))

temp = model.frame(fit_all_raw)[1:2,-1, drop=FALSE]
plot(fit_all_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(AU_raw) + ", round(coef(fit_all_raw)[1], 3), " * ENSO_raw + " , round(coef(fit_al

temp = model.frame(fit_AU_std)[1:2,-1, drop=FALSE]
plot(fit_AU_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(AU_std) + ", round(coef(fit_AU_std)[1], 3), " * ENSO_raw + " , round(coef(fit_AU_s
```

```
temp = model.frame(fit_all_std)[1:2,-1, drop=FALSE]
plot(fit_all_std, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(AU_std) + ", round(coef(fit_all_std)[1], 3), " * ENSO_std + " , round(coef(fit_al

temp = model.frame(fit_AU_raw)[1:2,-1, drop=FALSE]
plot(fit_AU_raw, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(AU_raw) + ", round(coef(fit_AU_raw)[1], 3), " * ENSO_std + " , round(coef(fit_AU_
```




***Interpretation:***

- When the predictors are standardized, it does not matter if we standardize the outcome or not. The coefficients are the same. The reason is, that only scale of the x-axis is changed (in the transformation function).
- Subtracting a constant from a predictor, changes the reference level.
- Dividing the predictor by a constant $\alpha$ changes the interpretation of $\beta$: for a $1 * \alpha$-unit-change in the predictor...
- When the predictors are standardized, it changes the interpretation of coefficients; the new reference level is the mean of the predictor and the coefficient indicates the change in log-odds for a 1 standard deviation change of the predictor.

AU = IOD

```
coef(Colr(au_raw ~iod_raw, order=len_theta))
```

```
##  iod_raw
## 0.889924
```

```
coef(Colr(AU_std ~iod_raw, order=len_theta))
```

```
##   iod_raw
## 0.8895465
```

```
coef(Colr(AU_std ~IOD_std, order=len_theta))
```

```
##   IOD_std
## 0.3807749
```

```
coef(Colr(au_raw ~IOD_std, order=len_theta))
```

```
##   IOD_std
## 0.3810426
```

AU = ENSO

```
coef(Colr(au_raw ~ enso_raw, order=len_theta))
```

```
##  enso_raw
## 0.3450753
```

```
coef(Colr(AU_std ~ enso_raw, order=len_theta))
```

```
##  enso_raw
## 0.6655537
```

```
coef(Colr(AU_std ~ ENSO_std, order=len_theta))
```

```
## ENSO_std
## 0.745476
```

```
coef(Colr(au_raw ~ ENSO_std, order=len_theta))
```

```
##  ENSO_std
## 0.7513855
```

As we can see, when enso is standardized, it does not matter much, if the outcome (AU) is standardized or not. The reason is that not the raw outcome is modelled but instead the log-odds of the outcome, and the baseline log-odds are represented by the baseline trafo. It transforms the outcome (regardless if standardized or raw) on the log-odds scale. Surprisingly, if raw Enso is taken, then the coefficients differ for raw or standardized Outcome. Enso has mean at 26 with standard deviation of around 1. Therefore, the reference point is Enso=0 and the coefficient indicates the change in log-odds for a 1 unit Enso increase. But because the mean is around 26, this is not a good reference point. I assume this is the reason why the first 2 coeffs differ.

AU = ENSO - mean(ENSO)

```r
enso_0mean <- enso_raw - mean(enso_raw)
coef(Colr(au_raw ~ enso_0mean, order=len_theta))
```

```
## enso_0mean
##  0.7472076
```

```r
coef(Colr(AU_std ~ enso_0mean, order=len_theta))
```

```
## enso_0mean
##    0.74744
```

```r
coef(Colr(AU_std ~ ENSO_std, order=len_theta))
```

```
## ENSO_std
## 0.745476
```

```r
coef(Colr(au_raw ~ ENSO_std, order=len_theta))
```

```
##  ENSO_std
## 0.7513855
```

When we use the same example as before, but subtract the mean of enso, we get the same coefficients as when using the standardized enso. This means that the reference point is now at mean(enso) and not at 0 anymore. I assume that this makes the estimated coefficients more reliable. -> but keep in mind that changing the predictor also changes the trafo.

IOD = ENSO

```r
iod_sc <- iod_raw/sd(iod_raw)

coef(fit31 <- Colr(iod_raw ~ enso_raw, order=len_theta))
```

```
##  enso_raw
## -1.327031
```

```r
coef(fit32 <- Colr(IOD_std ~ enso_raw, order=len_theta))
```

```
##  enso_raw
## -1.203324
```

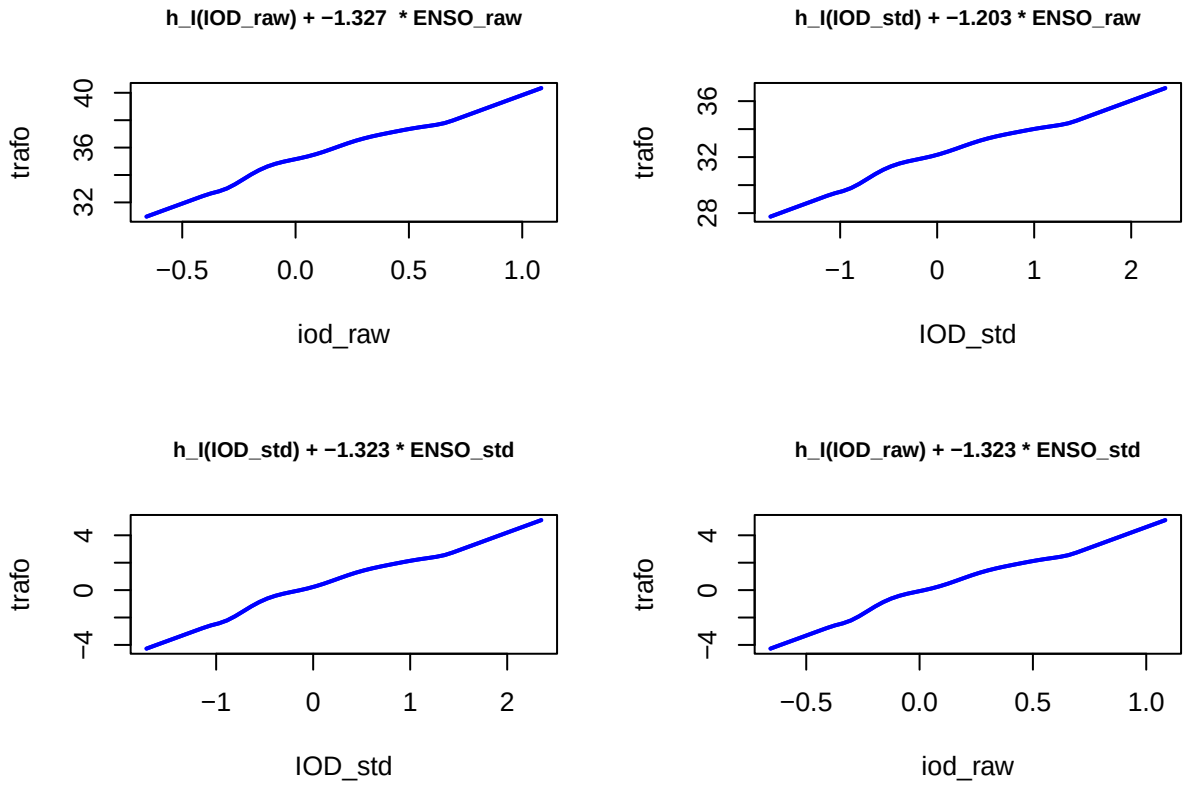
```r
# coef(fit32 <- Colr(iod_sc ~ enso_raw, order=len_theta))  # when only scaling IOD, same as raw!

coef(fit33 <- Colr(IOD_std ~ ENSO_std, order=len_theta))
```

```
##  ENSO_std
## -1.323209
```

```
coef(fit34 <- Colr(iod_raw ~ ENSO_std, order=len_theta))
```

```
##  ENSO_std
## -1.323099
```

```
par(mfrow = c(2,2))

temp = model.frame(fit31)[1:2,-1, drop=FALSE]
plot(fit31, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(IOD_raw) + ", round(coef(fit31)[1], 3), "  * ENSO_raw"), cex.main=0.8)

temp = model.frame(fit32)[1:2,-1, drop=FALSE]
plot(fit32, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(IOD_std) + ", round(coef(fit32)[1], 3),  " * ENSO_raw"), cex.main=0.8)

temp = model.frame(fit33)[1:2,-1, drop=FALSE]
plot(fit33, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(IOD_std) + ", round(coef(fit33)[1], 3),  " * ENSO_std"), cex.main=0.8)

temp = model.frame(fit34)[1:2,-1, drop=FALSE]
plot(fit34, which = 'baseline only', newdata = temp, lwd=2, col='blue',
     main=paste0("h_I(IOD_raw) + ", round(coef(fit34)[1], 3), " * ENSO_std"), cex.main=0.8)
```



```r
summary(iod_raw)
```

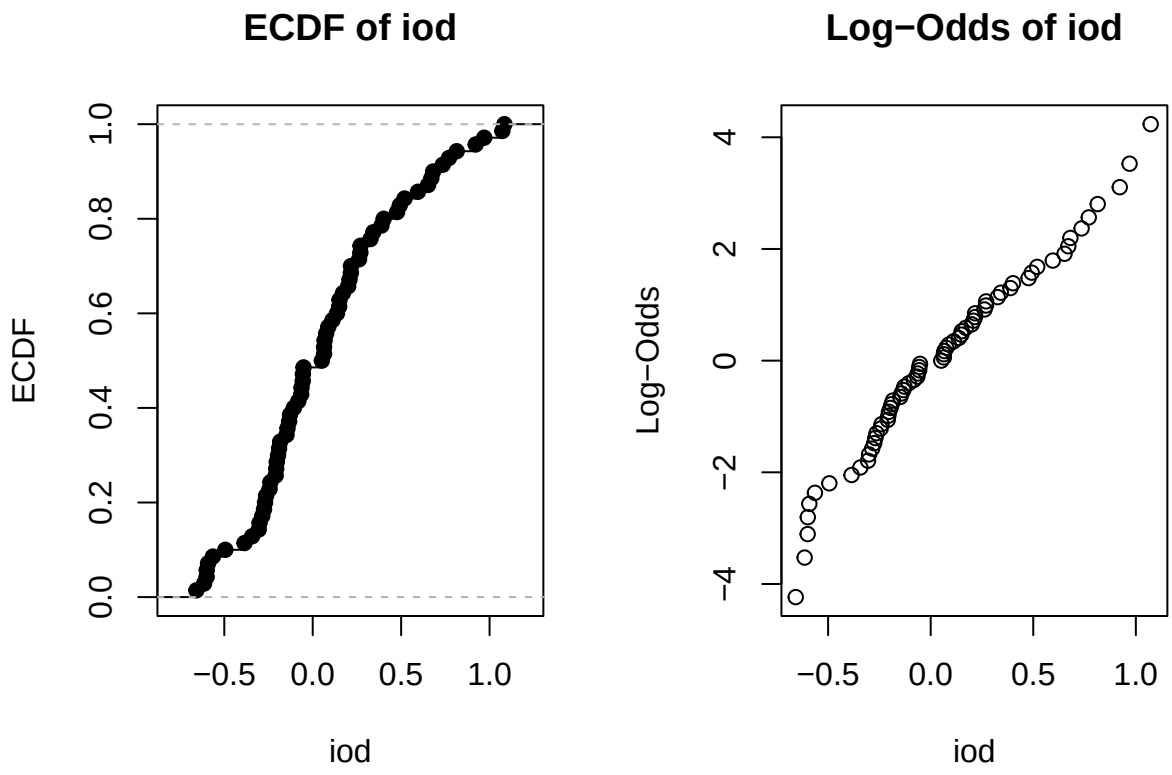
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.65786 -0.20857  0.05794  0.07765  0.31247  1.08397
```

```r
summary(au_raw)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 1.934e-06 3.945e-06 5.780e-06 6.095e-06 7.462e-06 1.594e-05
```

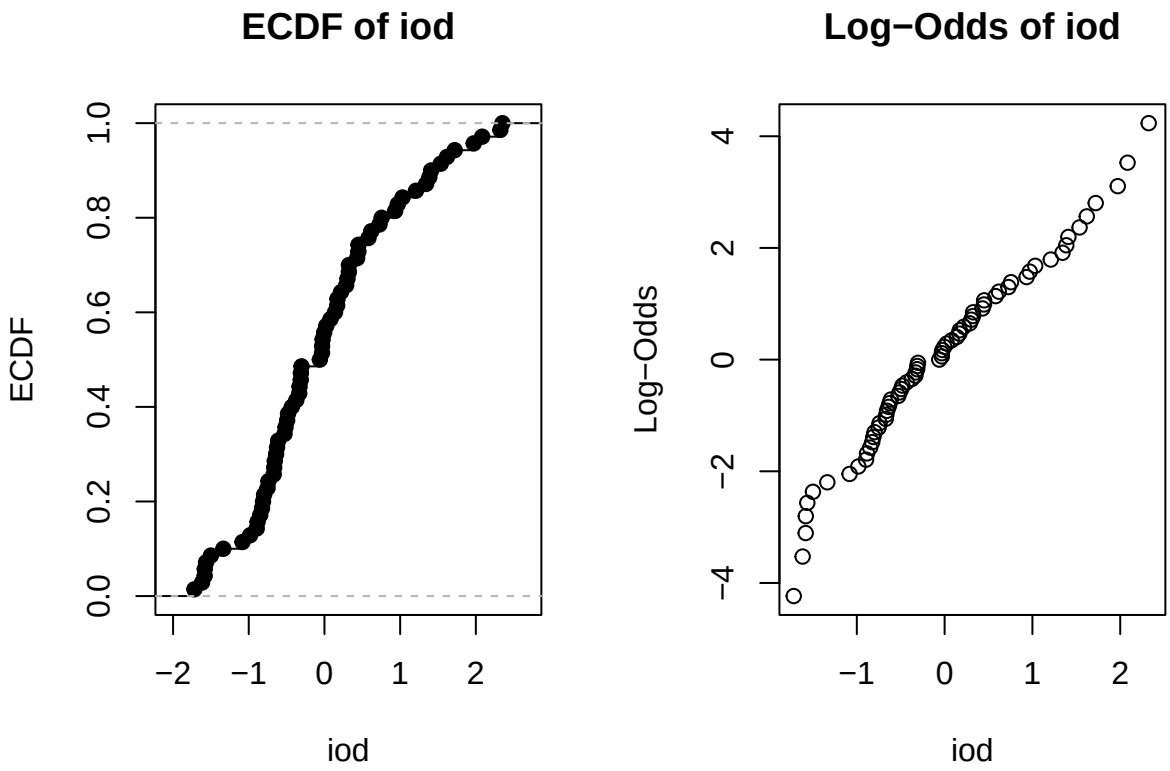
Log-odds of raw IOD:

```r
par(mfrow=c(1,2))
# ecdf of AU
ecdf_iod <- ecdf(iod_raw)
plot(ecdf_iod, main = "ECDF of iod", xlab = "iod", ylab = "ECDF")

# log-odds of iod
log_odds_iod <- log(ecdf_iod(iod_raw) / (1 - ecdf_iod(iod_raw)))
plot(iod_raw, log_odds_iod, main = "Log-Odds of iod", xlab = "iod", ylab = "Log-Odds")
```




Log-Odds of standardized AU:

```r
par(mfrow=c(1,2))

# ecdf of AU
ecdf_iod_std <- ecdf(IOD_std)
plot(ecdf_iod_std, main = "ECDF of iod", xlab = "iod", ylab = "ECDF")

# log-odds of iod
log_odds_iod_std <- log(ecdf_iod_std(IOD_std) / (1 - ecdf_iod_std(IOD_std)))
plot(IOD_std, log_odds_iod_std, main = "Log-Odds of iod", xlab = "iod", ylab = "Log-Odds")
```




The log-odds of iod and iod_std are the same, because log-odds only depend on the shape of the ECDF and are therefore not affected by scaling.

```r
log_odds_iod_std == log_odds_iod
```

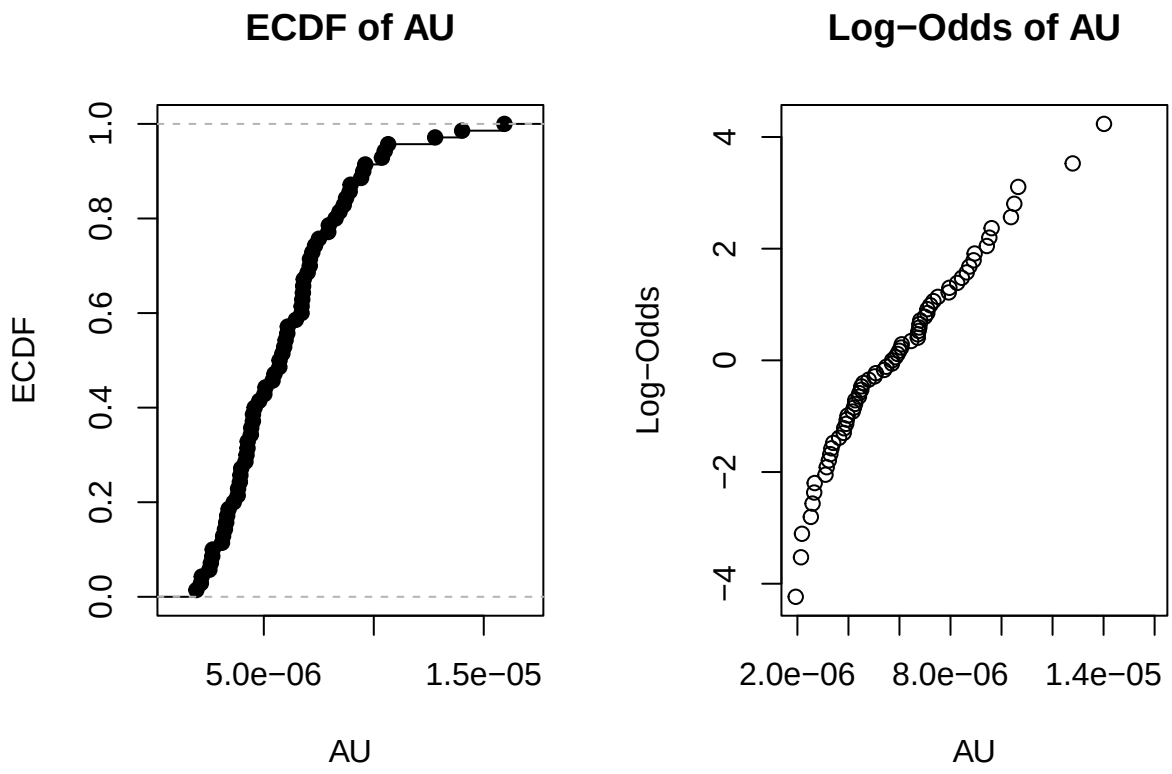
```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```r
# Colr(IOD_std ~ ENSO_std, order=len_theta)
# Colr(iod_raw ~ ENSO_std, order=len_theta)
#
```

```
# Colr(IOD_std ~ enso_raw, order=len_theta)
# Colr(iod_raw ~ enso_raw, order=len_theta)
```

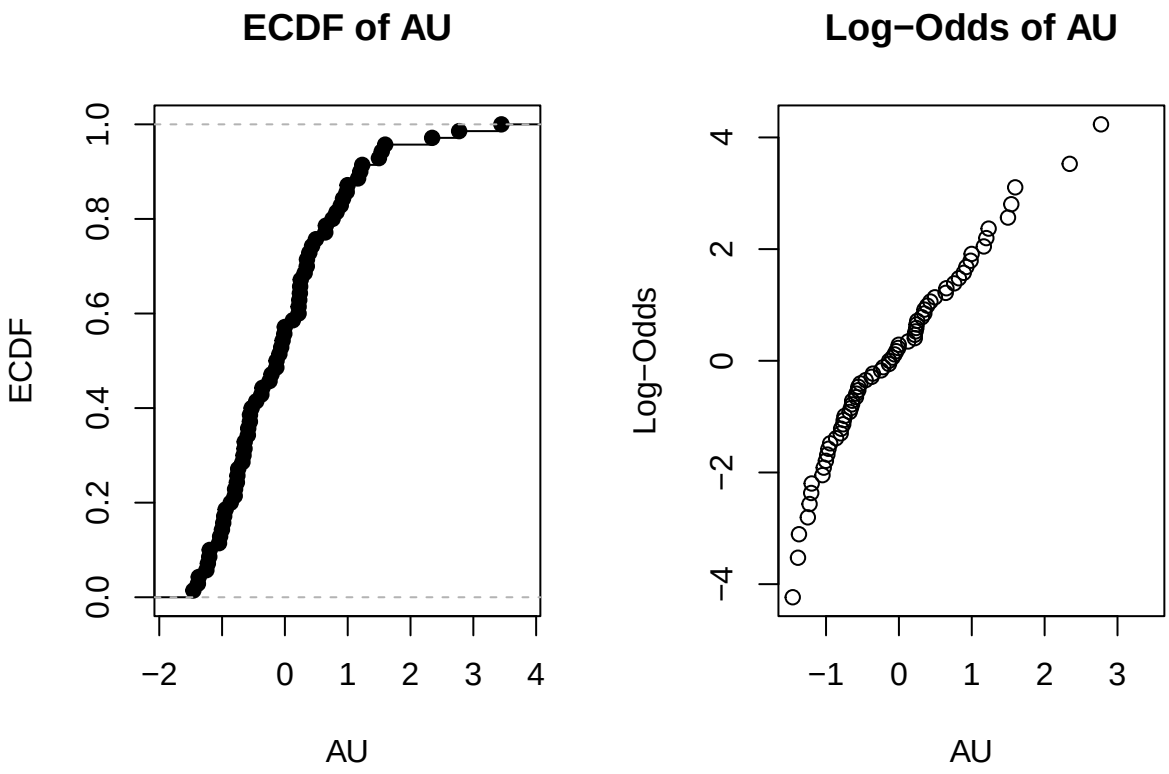
Log-odds of raw AU:

```
par(mfrow=c(1,2))
# ecdf of AU
ecdf_AU <- ecdf(au_raw)
plot(ecdf_AU, main = "ECDF of AU", xlab = "AU", ylab = "ECDF")

# log-odds of AU
log_odds_AU <- log(ecdf_AU(au_raw) / (1 - ecdf_AU(au_raw)))
plot(au_raw, log_odds_AU, main = "Log-Odds of AU", xlab = "AU", ylab = "Log-Odds")
```




Log-Odds of standardized AU:

```
par(mfrow=c(1,2))

# ecdf of AU
ecdf_AU_std <- ecdf(AU_std)
plot(ecdf_AU_std, main = "ECDF of AU", xlab = "AU", ylab = "ECDF")

# log-odds of AU
log_odds_AU_std <- log(ecdf_AU_std(AU_std) / (1 - ecdf_AU_std(AU_std)))
plot(AU_std, log_odds_AU_std, main = "Log-Odds of AU", xlab = "AU", ylab = "Log-Odds")
```




The log-odds of AU and AU_std are the same, because log-odds only depend on the shape of the ECDF and are therefore not affected by scaling.

```
log_odds_AU_std == log_odds_AU
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```