More Simulations

Harpeth Lee

3/29/2022

```r
library(ddsPLS2)
library(MASS)
library(pls)
library(mixOmics)
library(glmnet)
```

New Data Simulation Method

```r
sim_data <- function(n = 25, p = 50, q = 5, R = 5, noise_weight = 1, D_method = "newest", noise_type =

  # Creates A and D matrices

  Row1 <- c(rep(1, 5), rep(0, p - 5))
  Row2 <- c(rep(0, 5), rep(1, 10), rep(0, p - 15))

  reps1 <- round(3*R/5)
  reps2 <- R - reps1

  A1 <- do.call("rbind", replicate(3, Row1, simplify = FALSE))
  A2 <- do.call("rbind", replicate(2, Row2, simplify = FALSE))

  A <- rbind(A1, A2)

  if(struc == "complex") {
    R = 13

    Row3 <- c(rep(0, 15), rep(1, 5), rep(0, p - 20))
    Row4 <- c(rep(0, 20), 1, rep(0, p - 21))
    Row5 <- c(rep(0, 21), rep(1, 2), rep(0, p - 23))

    A3 <- do.call("rbind", replicate(3, Row3, simplify = FALSE))
    A4 <- do.call("rbind", replicate(2, Row4, simplify = FALSE))
    A5 <- do.call("rbind", replicate(3, Row5, simplify = FALSE))

    A <- rbind(A, A3, A4, A5)

  }


  if(D_method == "new") {
     D <- matrix(rep(1, R*q), nrow = R)
  } else if(D_method == "diag") {
    D <- diag(max(q, R))[1:R, 1:q]
  } else {
     q_s <- round(q/4)

     Row1D <- c(rep(1, q_s), rep(0, q - q_s))
     Row2D <- c(rep(0, q_s), rep(1, q_s), rep(0, q - 2*q_s))

     reps1D <- round(3*R/5)
     reps2D <- R - reps1D

     D1 <- do.call("rbind", replicate(reps1D, Row1D, simplify = FALSE))
     D2 <- do.call("rbind", replicate(reps2D, Row2D, simplify = FALSE))

     D <- rbind(D1, D2)
  }

  d <- ncol(A)+nrow(A)+ncol(D)
  psi <- MASS::mvrnorm(n = n,mu = rep(0,d),Sigma = diag(d))
  phi <- psi[,1:nrow(A)]

  phi <- matrix(rnorm(n*R), nrow = n)


  # If `rnorm` is used to generate noise a lower noise weight should be used as
  # the function is more sensitive since we directly weight results and not the
  # covariance matrix.

  if(noise_type == "mvrnorm") {
    epsilon_X <- mvrnorm(n = dim(phi)[1],
                         rep(0, dim(A)[2]),
                         Sigma = noise_weight*diag(dim(A)[2]))

    epsilon_Y <- mvrnorm(n = dim(phi)[1],
                         rep(0, dim(D)[2]),
                         Sigma = noise_weight*diag(dim(D)[2]))
  } else {
   epsilon_X <- matrix(noise_weight*rnorm(n = n*p), nrow = n)
   epsilon_Y <- matrix(noise_weight*rnorm(n = n*q), nrow = n)
  }

  X <- phi %*% A + epsilon_X
  Y <- phi %*% D + epsilon_Y

  #X <- scale(X)
  #Y <- scale(Y)

  list(X=X, Y=Y)
}
```

```
noise_rmse <- function(noise_weight, func = "ddsPLS"){
   sim <- sim_data(n = 300, p = 100, q = 5, noise_weight = noise_weight, noise_type = "rnorm", struc =
```

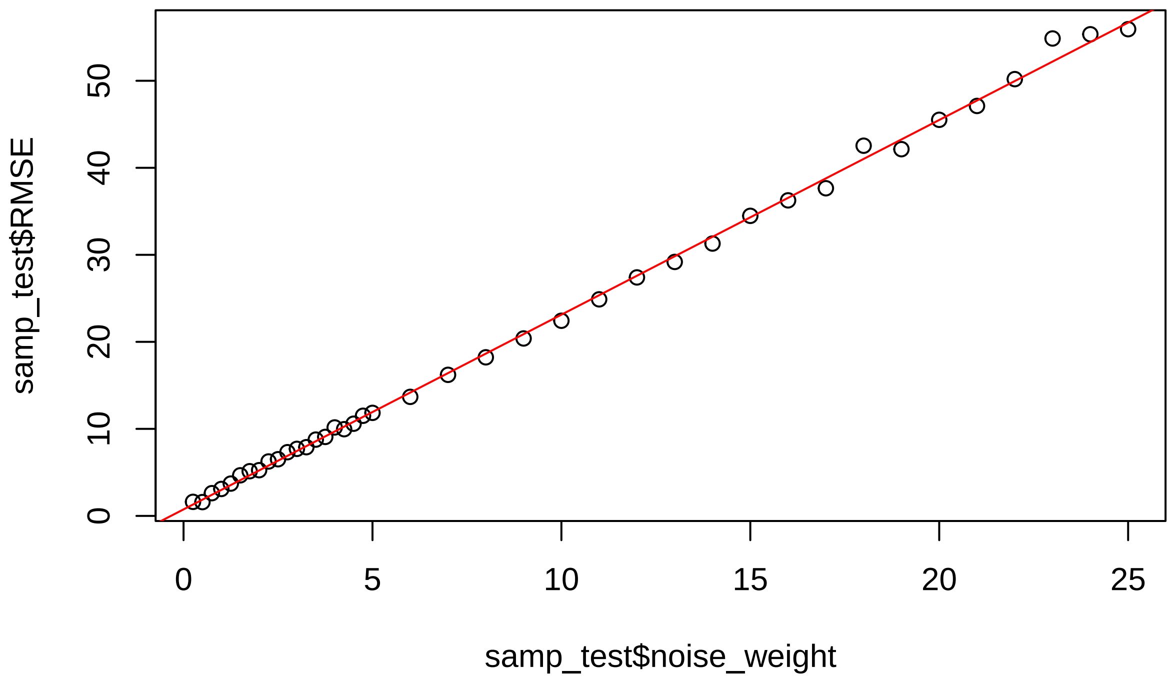
```r
  split <- sample(c(rep(0, 100), rep(1, 200)))

  sim_train_X <- sim$X[split == 0, ]
  sim_train_Y <- sim$Y[split == 0, ]

  sim_test_X <- sim$X[split == 1, ]
  sim_test_Y <- sim$Y[split == 1, ]

  # Generates model using the training set and predicts RMSE
  if(func == "ddsPLS") {
    mod <- ddsPLS(sim_train_X, sim_train_Y)

    preds <- predict(mod, sim_test_X)
    preds_ib <- predict(mod, sim_train_X)

    rmse <- sqrt(sum((preds$y_est - sim_test_Y)^2)/nrow(sim_test_Y))

    pred_mean_tr <- t(replicate(nrow(sim_train_Y), colMeans(sim_train_Y)))
    R2 <- 1 - (sum((preds_ib$y_est - sim_train_Y)^2)/sum((sim_train_Y - pred_mean_tr)^2))

    pred_mean_ts <- t(replicate(nrow(sim_test_Y), colMeans(sim_test_Y)))
    Q2 <- 1 - (sum((preds$y_est - sim_test_Y)^2)/sum((sim_test_Y - pred_mean_ts)^2))

    ncomp <- mod$R

  }
  else if(func == "lasso") {
    mod <- cv.glmnet(sim_train_X, sim_train_Y, family = "mgaussian")

    preds <- predict(mod, sim_test_X)[,,1]
    preds_ib <- predict(mod, sim_train_X)[,,1]

    rmse <- sqrt(sum((preds - sim_test_Y)^2)/nrow(sim_test_Y))

    pred_mean_tr <- t(replicate(nrow(sim_train_Y), colMeans(sim_train_Y)))
    R2 <- 1 - (sum((preds_ib - sim_train_Y)^2)/sum((sim_train_Y - pred_mean_tr)^2))

    pred_mean_ts <- t(replicate(nrow(sim_test_Y), colMeans(sim_test_Y)))
    Q2 <- 1 - (sum((preds - sim_test_Y)^2)/sum((sim_test_Y - pred_mean_ts)^2))

    ncomp <- NA

  } else {
    colnames(sim_train_X) <- c(1:100)
    colnames(sim_test_X) <- c(1:100)
    colnames(sim_test_Y) <- c(1:5)
    colnames(sim_train_Y) <- c(1:5)

    mod <- spls(sim_train_X, sim_train_Y, ncomp = 10)

    perf <- perf(mod, validation = "Mfold", folds = 10)
    Q2 <- perf$measures$Q2.total$values$value
    ncomp <- which(Q2 <= 0)[1]
```

```
      if(ncomp !=1) {
        ncomp <- ncomp - 1
      }

      preds <- predict(mod, sim_test_X)
      preds <- preds$predict[,,ncomp]

      preds_ib <- predict(mod, sim_train_X)
      preds_ib <- preds_ib$predict[,,ncomp]

      rmse <- sqrt(sum((preds - sim_test_Y)^2)/nrow(sim_test_Y))

      pred_mean_tr <- t(replicate(nrow(sim_train_Y), colMeans(sim_train_Y)))
      R2 <- 1 - (sum((preds_ib - sim_train_Y)^2)/sum((sim_train_Y - pred_mean_tr)^2))

      pred_mean_ts <- t(replicate(nrow(sim_test_Y), colMeans(sim_test_Y)))
      Q2 <- 1 - (sum((preds - sim_test_Y)^2)/sum((sim_test_Y - pred_mean_ts)^2))
    }

    out <- c(noise_weight, ncomp, rmse, R2, Q2, R2-Q2)

    return(out)
}
```

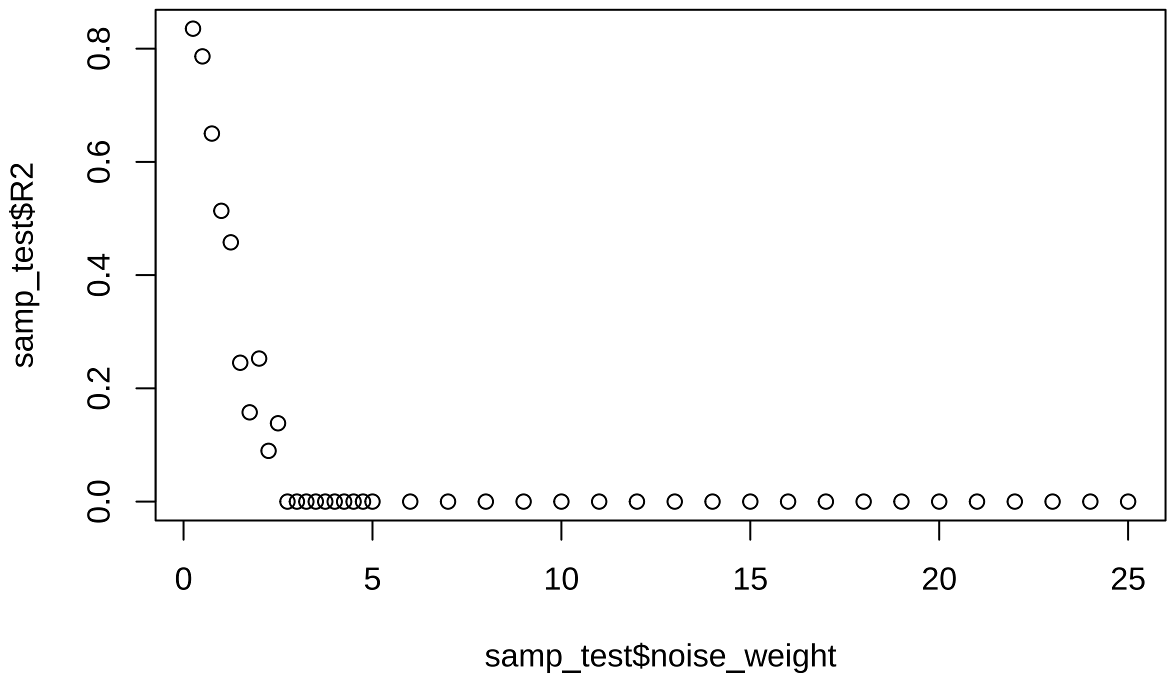
```
samp_test <- apply(matrix(c(seq(from = 0.25, to = 5, by = 0.25),
                            seq(from = 6, to = 25, by = 1)),
                            nrow = 1),
                   MARGIN = 2,
                   noise_rmse)
```

```
samp_test <-as.data.frame(t(samp_test))
colnames(samp_test) <- c("noise_weight", "ncomp", "RMSE", "R2", "Q2", "R2-Q2")

reg <- lm(RMSE ~ noise_weight, data = samp_test)

plot(samp_test$noise_weight, samp_test$RMSE)
abline(reg, col = "red")
```

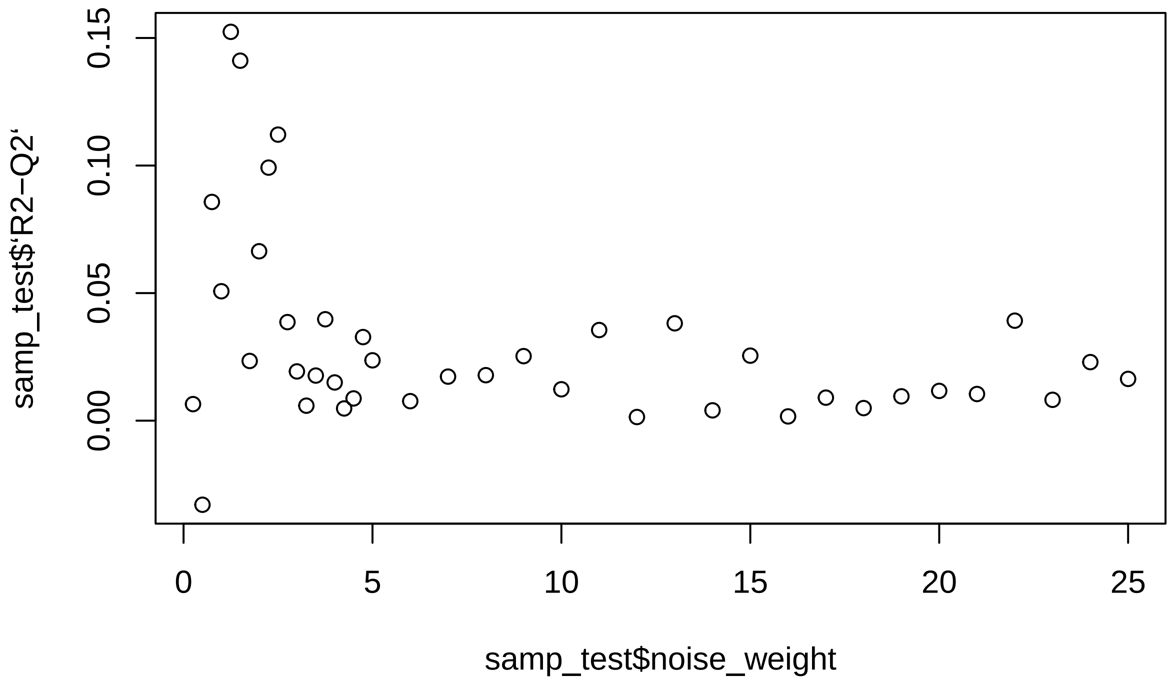
New Simulation Noise Test



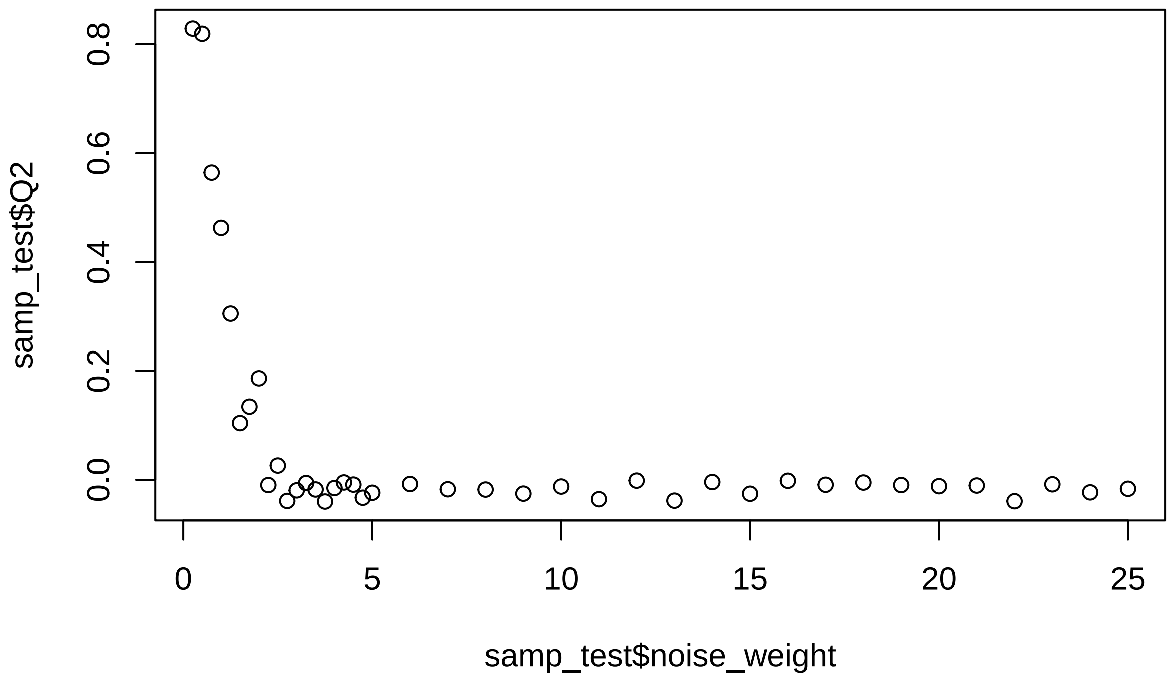
```r
plot(samp_test$noise_weight, samp_test$R2)
```



```r
plot(samp_test$noise_weight, samp_test$`R2-Q2`)
```



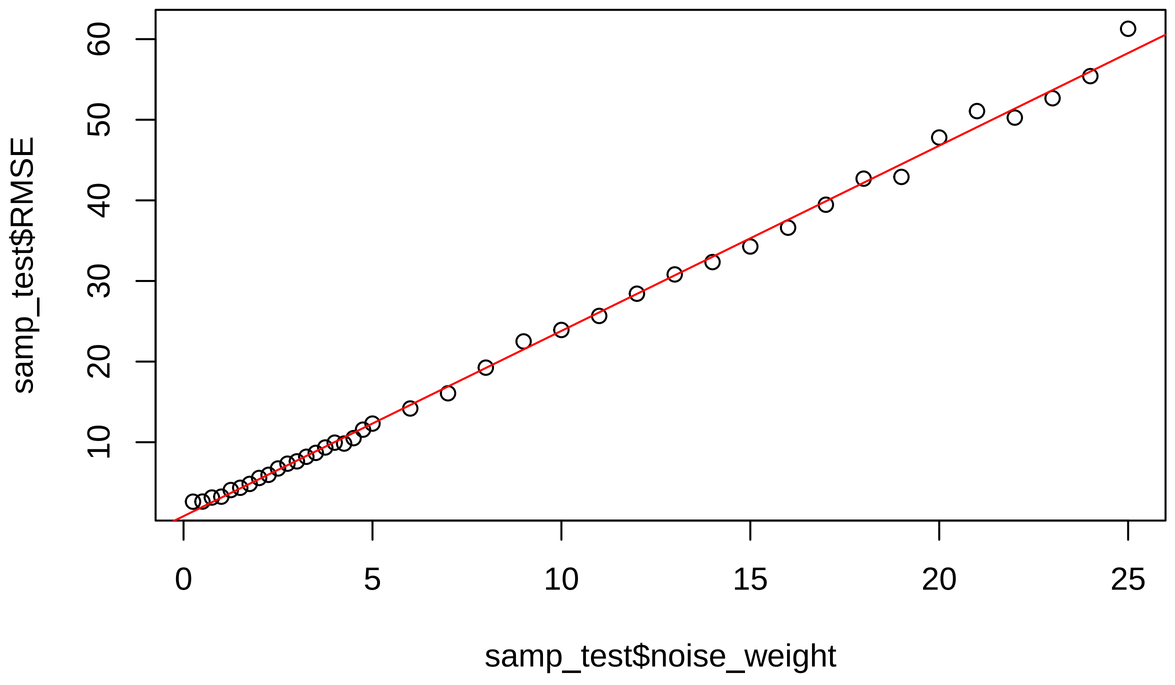
```
plot(samp_test$noise_weight, samp_test$Q2)
```



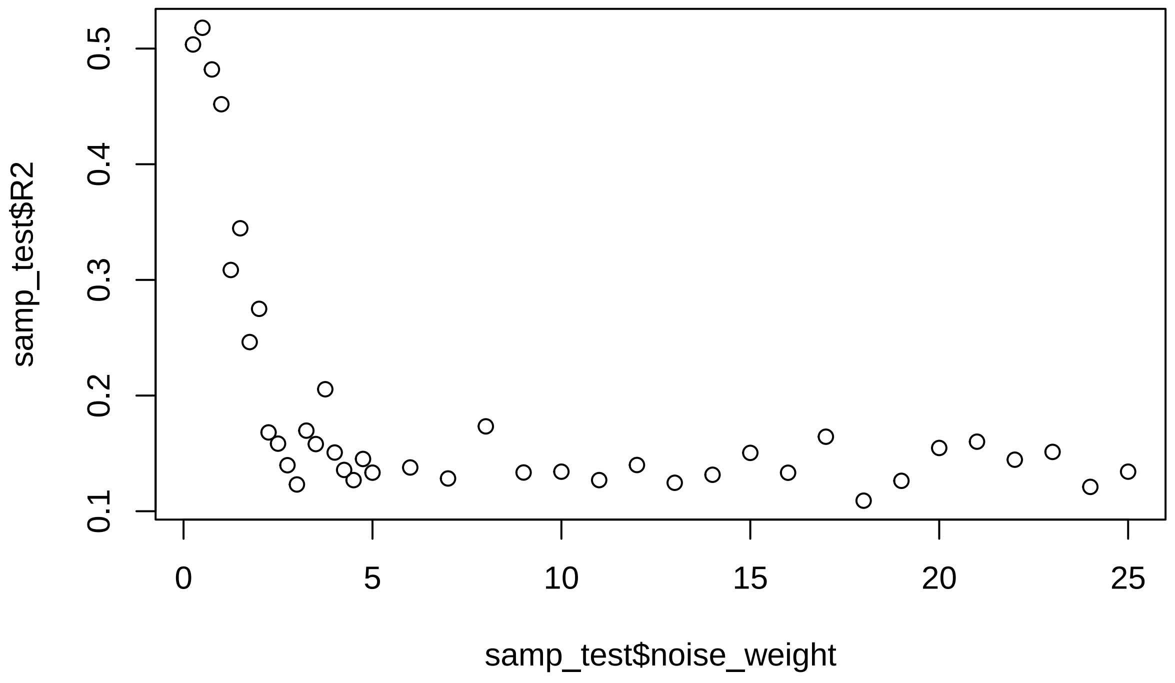
```
samp_test <- apply(matrix(c(seq(from = 0.25, to = 5, by = 0.25),
                            seq(from = 6, to = 25, by = 1)),
                            nrow = 1),
                   MARGIN = 2,
                   noise_rmse,
                   func = "spls")
```

```
samp_test <-as.data.frame(t(samp_test))
colnames(samp_test) <- c("noise_weight", "ncomp", "RMSE", "R2", "Q2", "R2-Q2")
```

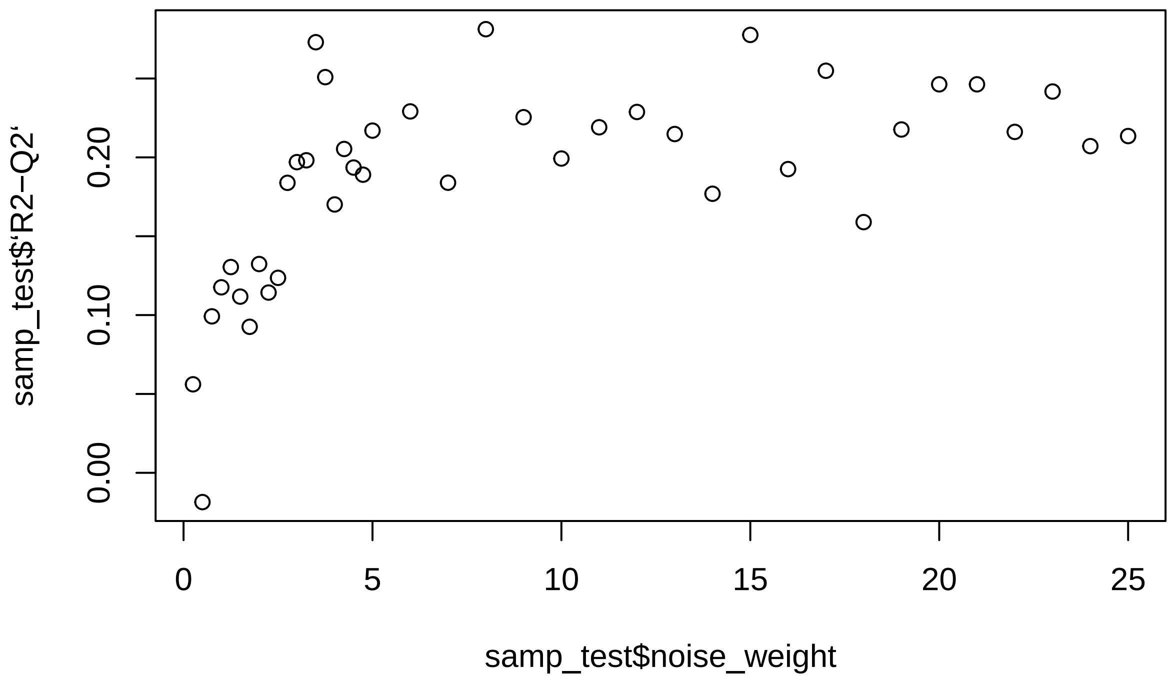
```r
reg <- lm(RMSE ~ noise_weight, data = samp_test)

plot(samp_test$noise_weight, samp_test$RMSE)
abline(reg, col = "red")
```



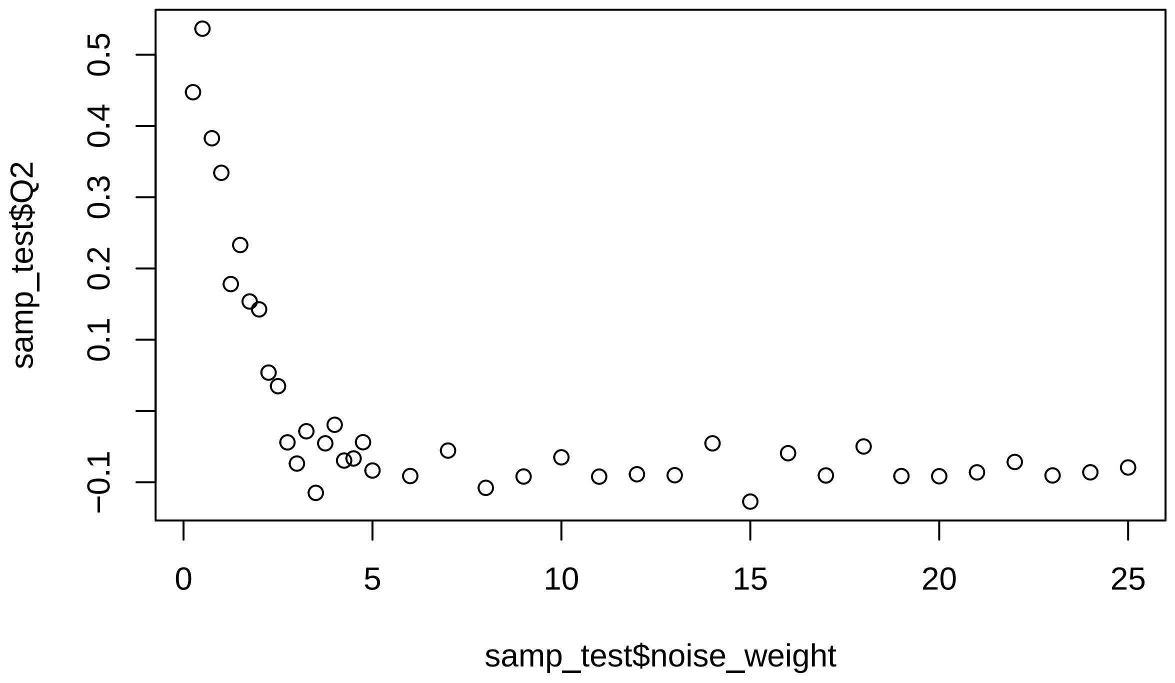
```r
plot(samp_test$noise_weight, samp_test$R2)
```



```r
plot(samp_test$noise_weight, samp_test$`R2-Q2`)
```



```
plot(samp_test$noise_weight, samp_test$Q2)
```



I am not sure if adding the results of each dimension works the best here, but it's what I have for now.

```
p_rmse <- function(p, noise_weight = 1, n = 150, func = "ddsPLS", struc = "complex"){

  # Randomly simulates data
  sim <- sim_data(n = n, p = p, q = 5, noise_weight = noise_weight, noise_type = "rnorm", struc = struc

  # Splits into training and test
```

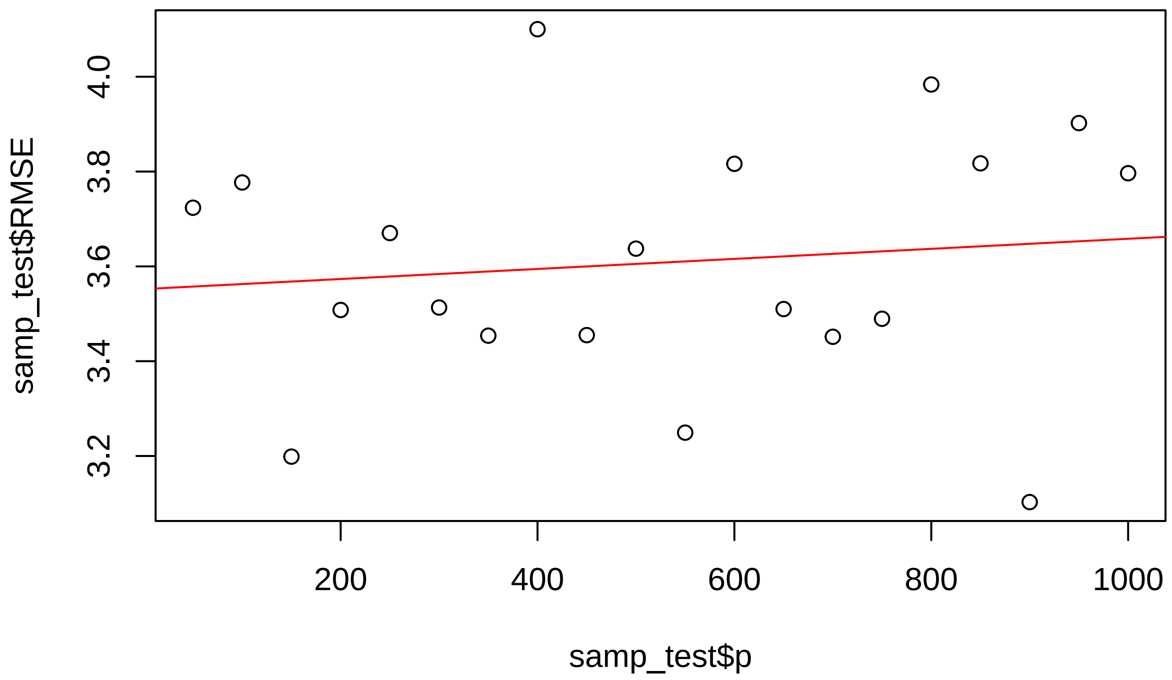
```
in_train <- round(n/3)
in_test <- round(2*n/3)

split <- sample(c(rep(0, in_train), rep(1, in_test)))

sim_train_X <- sim$X[split == 0, ]
sim_train_Y <- sim$Y[split == 0, ]

sim_test_X <- sim$X[split == 1, ]
sim_test_Y <- sim$Y[split == 1, ]
# Generates model using the training set and predicts RMSE
if(func == "ddsPLS") {
  mod <- ddsPLS(sim_train_X, sim_train_Y)

  preds <- predict(mod, sim_test_X)
  preds_ib <- predict(mod, sim_train_X)

  rmse <- sqrt(sum((preds$y_est - sim_test_Y)^2)/nrow(sim_test_Y))

  pred_mean_tr <- t(replicate(nrow(sim_train_Y), colMeans(sim_train_Y)))
  R2 <- 1 - (sum((preds_ib$y_est - sim_train_Y)^2)/sum((sim_train_Y - pred_mean_tr)^2))

  pred_mean_ts <- t(replicate(nrow(sim_test_Y), colMeans(sim_test_Y)))
  Q2 <- 1 - (sum((preds$y_est - sim_test_Y)^2)/sum((sim_test_Y - pred_mean_ts)^2))

  ncomp <- mod$R

}
else if(func == "lasso") {
  mod <- cv.glmnet(sim_train_X, sim_train_Y, family = "mgaussian")

  preds <- predict(mod, sim_test_X)[,,1]
  preds_ib <- predict(mod, sim_train_X)[,,1]

  rmse <- sqrt(sum((preds - sim_test_Y)^2)/nrow(sim_test_Y))

  pred_mean_tr <- t(replicate(nrow(sim_train_Y), colMeans(sim_train_Y)))
  R2 <- 1 - (sum((preds_ib - sim_train_Y)^2)/sum((sim_train_Y - pred_mean_tr)^2))

  pred_mean_ts <- t(replicate(nrow(sim_test_Y), colMeans(sim_test_Y)))
  Q2 <- 1 - (sum((preds - sim_test_Y)^2)/sum((sim_test_Y - pred_mean_ts)^2))

  ncomp <- NA

} else {
  colnames(sim_train_X) <- c(1:ncol(sim_train_X))
  colnames(sim_test_X) <- c(1:ncol(sim_train_X))
  colnames(sim_test_Y) <- c(1:ncol(sim_test_Y))
  colnames(sim_train_Y) <- c(1:ncol(sim_test_Y))

  mod <- spls(sim_train_X, sim_train_Y, ncomp = 10)

  perf <- perf(mod, validation = "Mfold", folds = 10)
```

```r
      Q2 <- perf$measures$Q2.total$values$value
      ncomp <- which(Q2 <= 0)[1]
      if(ncomp !=1) {
        ncomp <- ncomp - 1
      }

      preds <- predict(mod, sim_test_X)
      preds <- preds$predict[,,ncomp]

      preds_ib <- predict(mod, sim_train_X)
      preds_ib <- preds_ib$predict[,,ncomp]

      rmse <- sqrt(sum((preds - sim_test_Y)^2)/nrow(sim_test_Y))

      pred_mean_tr <- t(replicate(nrow(sim_train_Y), colMeans(sim_train_Y)))
      R2 <- 1 - (sum((preds_ib - sim_train_Y)^2)/sum((sim_train_Y - pred_mean_tr)^2))

      pred_mean_ts <- t(replicate(nrow(sim_test_Y), colMeans(sim_test_Y)))
      Q2 <- 1 - (sum((preds - sim_test_Y)^2)/sum((sim_test_Y - pred_mean_ts)^2))
    }

    out <- c(p, ncomp, rmse, R2, Q2, R2-Q2)

    return(out)
}
```

```r
samp_test <- apply(matrix(seq(from = 50, to = 1000, by = 50),
                          nrow = 1),
                   MARGIN = 2,
                   p_rmse)
```

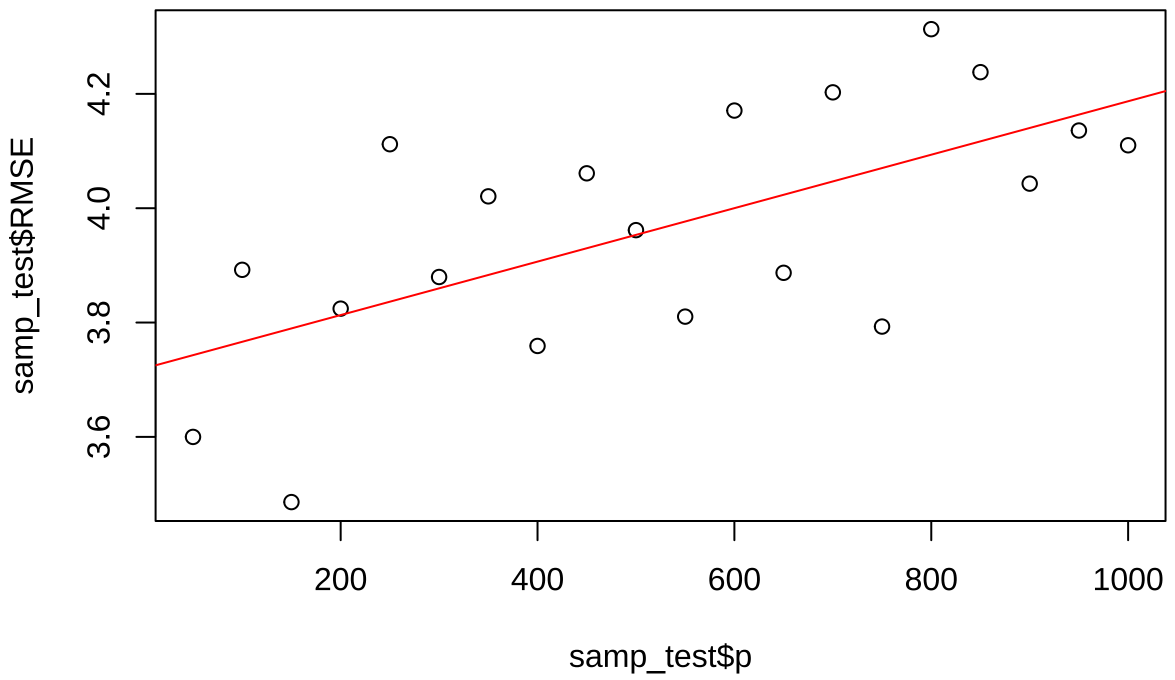
```r
samp_test <-as.data.frame(t(samp_test))
colnames(samp_test) <- c("p", "ncomp", "RMSE", "R2", "Q2", "R2-Q2")

reg <- lm(RMSE ~ p, data = samp_test)

plot(samp_test$p, samp_test$RMSE)
abline(reg, col = "red")
```

New Simulation Predictors Test



```r
samp_test <- apply(matrix(seq(from = 50, to = 1000, by = 50),
                          nrow = 1),
                   MARGIN = 2,
                   p_rmse,
                   func = "spls")
```

```r
samp_test <-as.data.frame(t(samp_test))
colnames(samp_test) <- c("p", "ncomp", "RMSE", "R2", "Q2", "R2-Q2")

reg <- lm(RMSE ~ p, data = samp_test)

plot(samp_test$p, samp_test$RMSE)
abline(reg, col = "red")
```



It looks like ddsPLS performs much better than sPLS when a large number of meaningless predictors are added. I need to figure out how to select the ideal number of components for sPLS models to use and how to calculate the predicted values. Run more simulations to see how model performs. Compare to LASSO. Comparing the number of components. Add LASSO to the theory section.

```
ddspls.p.reps <- replicate(100, p_rmse(p = 1000))
```

```
spls.p.reps <- replicate(100, p_rmse(p = 1000, func = "spls"))
```

```
ddspls.p.reps <- read.csv2("/Users/johnlee/R Files/thesis-lee/Simulations/data/ddspls.p.reps.csv")
spls.p.reps <- read.csv2("/Users/johnlee/R Files/thesis-lee/Simulations/data/spls.p.reps.csv")

ddspls.p.reps <- t(ddspls.p.reps)[-1, ]
spls.p.reps <- t(spls.p.reps)[-1, ]
```

```
n <- length(ddspls.p.reps[,2])

mn <- mean(ddspls.p.reps[,2])

sd <- sd(ddspls.p.reps[,2])

error <- qnorm(0.975)*sd/sqrt(n)

c(mn - error, mn + error)
```

```
## [1] 12.64318 17.46620
```

```
n <- length(spls.p.reps[,2])

mn <- mean(spls.p.reps[,2])

sd <- sd(spls.p.reps[,2])

error <- qnorm(0.975)*sd/sqrt(n)
```

```
c(mn - error, mn + error)
```

```
## [1] 12.06256 16.57762
```

It looks like there won't be that much difference in RMSE between the two for larger values of p.