Surrogate Modeling and Image Compression

Arjun Kakkar

12/15/2017

Abstract

The objective of this project is to improve a compression decompression algorithm using the tools of statistical design and optimiziation. This is done by constructing an optimal block design with the parameters of the procedure as factors. Thus we have two quantitative variables and one categorical variable and we have two measured responses, the quality of the image and the time taken for computation. Using the results from conducting this experiment, we create response surfaces that approximate both response values in the enitre parameter space. These responce surfaces are simultaneuosly optimized by combining them into a cost function and once optimum values are found, the optimized procedure is tested to quantify improvements. Overall it is found that for a particular cost function, the procedure results in a 20% improvement in quality and 27% increase in computational time over the unoptimized procedure.

PDE-based Image Compression

Image compression and more broadly signal recovery problems are of great interest due to their promised savings in space. Furthermore, data corruption and partial data loss are problems encountered frequently and methods to effectively 'recover' the data are important to consider. In different contexts one can make different assumptions about the nature of the underlying data in order to find the most optimal value to substitute the missing information with. In the context of PDE based image recovery, the underlying assumption is that the image satisfies some differential operator L such that

$$L(U) = 0,$$

where U is a matrix of pixel values. Therefore, given partial information in U one can achieve a reconstruction, roughly speaking, by just finding missing values that satisfy the above equation.

In our case specifically, we look at the differential operator $L = \nabla^2$, which corresponds to saying that missing information in the image satisfies the heat equation $\nabla^2 U = 0$. What this visibly corresponds to is smoothing the image out in areas that do not contain information.

The Decompression Procedure

The procedure for finding the solution given certain known values, however, is not trivial. A direct solution based on a system of linear equations reduces to a matrix inversion problem. The issue is that for most practically sized images, these matrices are huge and computing their inverse is often intractable. Instead we pursue an iterative approach by converting the problem into finding the steady state solution of the following equation in two dimensions

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}$$

Finally, the procedure in implementation is this. We first populate the matrix U with known values of the original image. Then we initialize all the unknown values randomly. We then apply the following iteration at each unknown value of U:

$$U_{t+\Delta t}(x, y) = U_t(x, y) + \delta(U_t(x, y+1) + U_t(x+1, y) + U_t(x, y-1) + U_t(x-1, y) - 4U_t(x, y)),$$

where $U_t(x, y)$ corresponds to the value of the pixel at location (x, y) and at time t of convergence procedure.

This iteration is repeated at every unknown pixel value until the size of the step from one iteration to the next decreases below a critical threshold value. That is when the execution is halted and the output U_T is considered to be the reconstructed image.

Output and Improvements

Once we generate the reconstructed image, we can calculate the quality of the reconstruction by comparing it to the original image and calculating a mean squared distance between the pixel values. We can also measure the time taken for the reconstruction to talk about the real world applicability of the procedure.

Now, in order to improve the performance of this procedure, we want to optimize these outputs such that we get the maximum quality reproduction in the least amount of time. This is exactly where the selection of the kind of information stored in the compression step becomes incredibly important. From preliminary experimentation it is increasingly clear that the choice of points at which to store infromation influences both quality and time. In fact, there seems to be a tradeoff between the amount of information stored (compression ratio), the proportion of that information stored at the edges of the image (proportion on edge) and the thickness of the edges of the image in pixels (thickness).

Design of Experiement

We begin by exploring the relationship between the three factors, compression, proportion on edge and thickness and the outputs, quality and time. Therefore we want to fit a response surface for each of the outputs and need a design that captures these relationships upto quadratic terms. Furthermore, we want to be able to make generalizable statements about all images and need to incorporate image to image variability in our design and the experiment that we conduct. We also have a somewhat artificial constraint of 100 runs due to computational time.

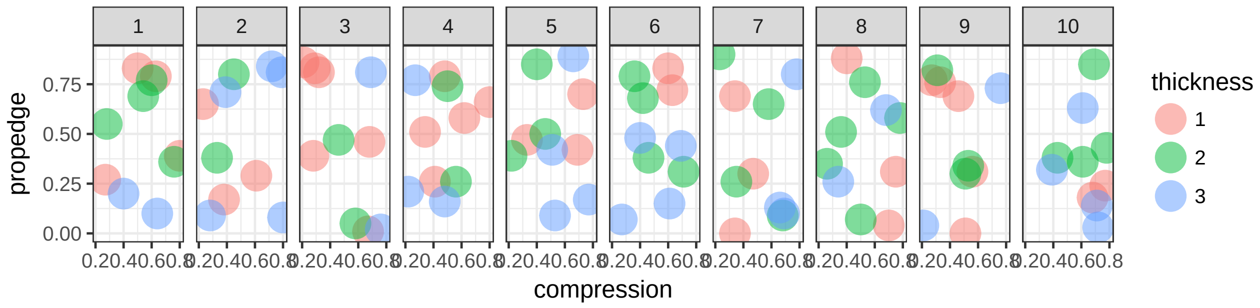
Given this information we can start making decisions about the kind of experiment we want to conduct. First we consider the feasible values for our factors under consideration. For compression ratio we determine that values under 0.2 give bad reproduction and would not see much practical use. Similarly, values above 0.8 do not give much of a size advantage over the original image. For proportion of pixels on the edges, we can have as little as 0, where there is no bias in selecting the initial pixels and can go all the way up to 0.9 where most of the initial information is stored on the edges. The value 0.9 is also chosen as a result of a failed experiment in which the upper limit was selected to be 1 and convergence of the solver was an issue. For thickness of the edge we chose values of $1, 2$ and 3 pixels since those had the largest effect during preliminary testing. We can summarize these considerations in the following table of the design region.

	Type	Values/Range
Compression Ratio	Quantitative	0.2-0.8
Proportion on Edge	Quantitative	0-0.9
Edge Thickness	Categorical	1, 2, 3

Table 1: Design Region

In order to incorporate image to image variability, we will use 10 different blocks that correspond to 10 different images randomly selected and conduct 10 runs on each of those images. We must now decide the settings at which to conduct these runs and use an optimal design strategy. We use optimal design over the traditional design methods due to the constraints on the number of runs and the number of blocks. The optimal design is created using the following code and the runs are visualized below.

```r
# Create candidates for optBlock to pick from
candidates = expand.grid(compression = seq(0.2, 0.8, 0.01),
                         propedge = seq(0, 0.9, 0.01),
                              thickness = factor(1:3))
# Create optimal design
optdes <- optBlock(~quad(compression, propedge)*thickness+thickness, candidates,
                   blocksizes = rep(10, 10), criterion = "Dpc", nRepeats = 1000)
# Plot of the design
design = mutate(optdes$design, block = c(rep(1, 10),rep(2, 10),rep(3, 10),rep(4, 10),
                                         rep(5, 10),rep(6, 10),rep(7, 10),rep(8, 10),
                                         rep(9, 10),rep(10, 10)))
ggplot(aes(x = compression, y = propedge, color = thickness), data = design) +
  geom_point(alpha = 0.5, size = 6)+
  facet_grid(.~block)+
  theme_bw()
```



NOTE: Since optBlock optimizes stochastically, every design obtained is different. Therefore the above design is only a representation of the design that was actually conducted. To see the actual design, refer to the MATLAB code in the appendix.

This design was then used to conduct runs on the 10 different randomly selected images and the results were imported into R (see MATLAB code).

```
# Actual design that was run
setwd("~/STAT-compression")
design <- read.csv('newdesign.csv', header = FALSE)

m <- mapply(c,design[,1:3], design[,4:6], design[,7:9], design[,10:12], design[,13:15],
            design[,16:18], design[,19:21], design[,22:24], design[,25:27], design[,28:30])
design <- data.frame(c(rep(1, 10), rep(2, 10), rep(3, 10), rep(4, 10), rep(5, 10),
                       rep(6, 10), rep(7, 10), rep(8, 10), rep(9, 10), rep(10, 10)), m)
colnames(design) <- c("block", "comp", "prop", "thick")
design <- design %>% mutate(block = as.factor(block), thick = as.factor(thick))

# Results from running the experiment
responses <- read.csv("newresults.csv", header = FALSE) %>%
  rename(mse = V1, time = V2)

# Complete dataframe for analysis
image_data <- cbind(design, responses)
```

Creation of Response Surfaces

Using the results of the experiment we can fit a model to estimate the effects of the different components. Note that since we have a random block effect and fixed effects for the other factors, we must fit a mixed effects model to the data. Furthermore, since we have two responses we must fit two different models. To do this we use the following code.

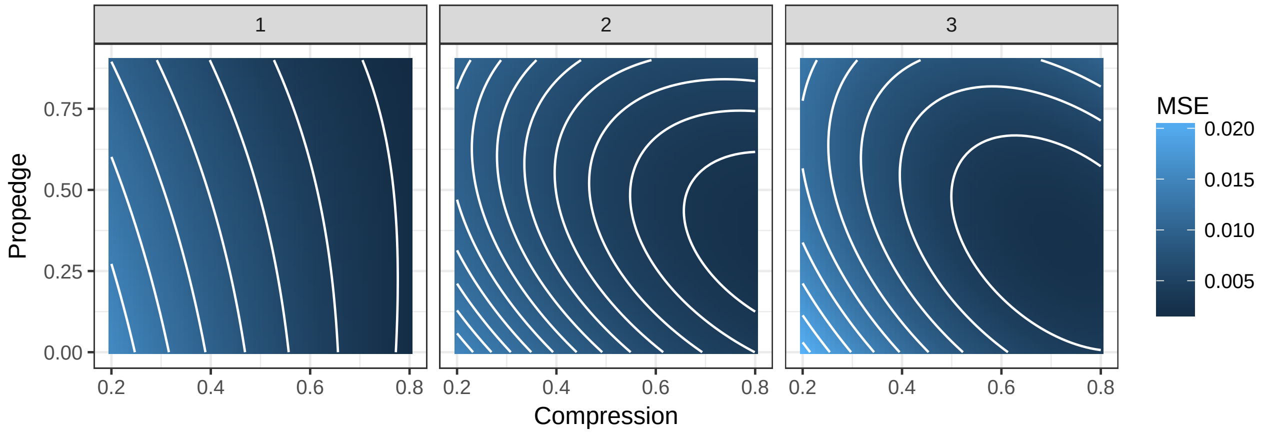
```r
# Model for MSE
mse_model <- lmer(mse~1 + (1|block) + SO(comp, prop):thick+thick, data = image_data)

# Model for time
time_model <- lmer(time~1 + (1|block) + SO(comp, prop):thick+thick, data = image_data)
```

Using the results of the fitted model, we can construct six response surfaces, one for each each level of thickness for MSE and time. This is done using the coefficients of the model as follows.

```r
# Store the model in equations for each level of MSE
msethick1 <- function(c, p) {
    val <- 0.0222961 - 0.0369953 * c - 0.0069362 * p + 0.009634 * p * c + 0.0138763 *
        c^2 - 0.0012087 * p^2
    return(val)
}
msethick2 <- function(c, p) {
    val <- 0.0222961 - 0.0009052 - 0.0359361 * c - 0.0188273 * p + 0.0115831 *
        p * c + 0.0177358 * c^2 + 0.0128854 * p^2
    return(val)
}
msethick3 <- function(c, p) {
    val <- 0.0222961 + 0.0108773 - 0.0703937 * c - 0.0319732 * p + 0.0254489 *
        p * c + 0.0425258 * c^2 + 0.0200453 * p^2
    return(val)
}
# Store the model in equations for each level of time
timethick1 <- function(c, p) {
    val <- 415.6468 - 824.9586 * c + 50.3482 * p + 147.771 * p * c + 426.3995 *
        c^2 - 101.8634 * p^2
    return(val)
}
timethick2 <- function(c, p) {
    val <- 415.6468 + 41.033 - 828.3122 * c - 63.8428 * p + 170.0446 * p * c +
        399.1249 * c^2 + 0.2417 * p^2
    return(val)
}
timethick3 <- function(c, p) {
    val <- 415.6468 + 258.3016 - 1965.1395 * c + 32.5666 * p - 305.0762 * p *
        c + 1693.4228 * c^2 + 173.8884 * p^2
    return(val)
}
# Plot contour maps of each response surface
cvals = seq(0.2, 0.8, 0.01)
pvals = seq(0, 0.9, 0.01)
# Plot contours for the response surface of MSE
msedata = data.frame(expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2],
    mapply(msethick1, expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2]),
    mapply(msethick2, expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2]),
    mapply(msethick3, expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2])) %>%
    gather(Type, Value, Var1.1, Var1.2, Var1.3) %>% mutate(Type = c(rep("1",
    5551), rep("2", 5551), rep("3", 5551)))
colnames(msedata) = c("Compression", "Propedge", "Type", "MSE")
```

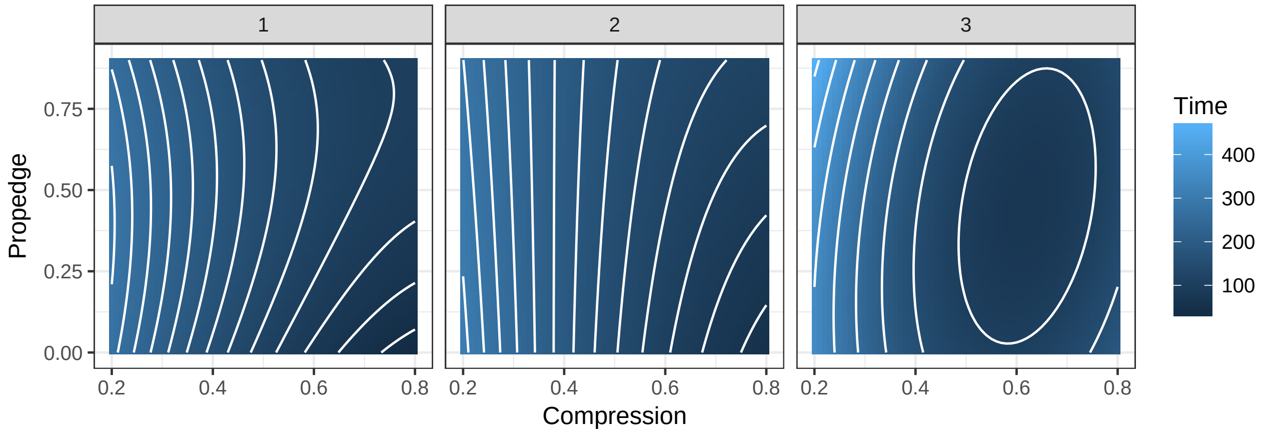
```
ggplot(msedata, aes(Compression, Propedge, z = MSE)) + geom_raster(aes(fill = MSE)) +
    geom_contour(colour = "white") + facet_grid(. ~ Type) + theme_bw() + labs(caption = "This plot shows
```



This plot shows the contours for the response surfaces of MSE.

```
# Plot contours for the response surface of time
timedata = data.frame(expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2],
    mapply(timethick1, expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2]),
    mapply(timethick2, expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2]),
    mapply(timethick3, expand.grid(cvals, pvals)[1], expand.grid(cvals, pvals)[2])) %>%
    gather(Type, Value, Var1.1, Var1.2, Var1.3) %>% mutate(Type = c(rep("1",
    5551), rep("2", 5551), rep("3", 5551)))
colnames(timedata) = c("Compression", "Propedge", "Type", "Time")

ggplot(timedata, aes(Compression, Propedge, z = Time)) + geom_raster(aes(fill = Time)) +
    geom_contour(colour = "white") + facet_grid(. ~ Type) + theme_bw() + labs(caption = "This plot shows
```



This plot shows the contours for the response surfaces of time.

Multi-Objective Optimization

Given these response surfaces, we can talk about the combination of values for which we obtain the lowest MSE and time. However, we want to consider the combined optimum. Therefore, we must combine the response surfaces for MSE and time into one response such that we can directly optimize for that response.

In order to do this we define a cost function such that any response of time above 240 seconds gets a large cost penalty and any MSE above 0.0125 gets a large cost penalty. Within those upper bounds, the cost function puts a similar emphasis on MSE and time. This function includes desirable characteristics of both good quality and fast compression and can be represented as

$$\begin{aligned} C(c,p,t) =& mse(c,p,t) + time(c,p,t)/20000 + \\ & I(mse(c,p,t) - 0.0125) \times (1 + mse(c,p,t))^{100} + \\ & I(time(c,p,t) - 240) \times (1 + time(c,p,t))^{100}, \end{aligned}$$

where $I(x)$ is an indicator function which is 1 if $x > 0$ and 0 otherwise.

We can implement this funcion in R and then given a compression ratio, we can find values for propedge and thickness such that the cost is minimum. Using this approach, we tabulate values of compression ratios and optimal settings for each ratio using the following code.

```
# Define a cost function for each different value of thickness
responsesurf1 <- function(inputs, c){
  p <- inputs[1]
  f1 = 0
  f2 = 0
  f3 = 0
  if(msethick1(c, p)>0.0125){
    f1 <- (msethick1(c, p)+1)^100
  }
  if (timethick1(c, p)>240){
    f2 <- (1+(timethick1(c, p)/20000))^100
  }
  f3 <- msethick1(c, p) + timethick1(c, p) / 20000
  return(f1+f2+f3)
}
responsesurf2 <- function(inputs, c){
  p <- inputs[1]
  f1 = 0
  f2 = 0
  f3 = 0
  if(msethick2(c, p)>0.0125){
    f1 <- (msethick2(c, p)+1)^100
  }
  if (timethick2(c, p)>240){
    f2 <- (1+(timethick2(c, p)/20000))^100
  }
  f3 <- msethick2(c, p) + timethick2(c, p) / 20000
  return(f1+f2+f3)
}
responsesurf3 <- function(inputs, c){
  p <- inputs[1]
  f1 = 0
  f2 = 0
  f3 = 0
  if(msethick3(c, p)>0.0125){
    f1 <- (msethick3(c, p)+1)^100
  }
  if (timethick3(c, p)>240){
    f2 <- (1+(timethick3(c, p)/20000))^100
  }
  f3 <- msethick3(c, p) + timethick3(c, p) / 20000
  return(f1+f2+f3)
}

# Compute optimal values and store in a dataframe
out = NULL
for(i in 1:101){
  cnow = 0.2 + (i-1)*(0.6/100)
  opt1 <- optimize(responsesurf1, c(0,0.9), c = cnow)
  opt2 <- optimize(responsesurf2, c(0,0.9), c = cnow)
  opt3 <- optimize(responsesurf3, c(0,0.9), c = cnow)
  minima <- rbind(as.data.frame(opt1),as.data.frame(opt2),as.data.frame(opt3))
  best <- which.min(minima$objective)
  mincost <- minima[best, 2]
  optp <- minima[best, 1]
  optmse <- switch (best, msethick1(cnow, optp), msethick2(cnow, optp), msethick3(cnow, optp))
  opttime <- switch (best, timethick1(cnow, optp), timethick2(cnow, optp), timethick3(cnow, optp))
  out = rbind(out, c(cnow, optp, best, optmse, opttime, mincost))
}
optimumsettings <- data.frame(out)
colnames(optimumsettings) = c("comp","propedge","thick","msepred","timepred","cost")
head(optimumsettings) %>% xtable(caption = "Optimum settings for different compression ratios with predi
```

	comp	propedge	thick	msepred	timepred	cost
1	0.20	0.90	1.00	0.01	257.12	3.61
2	0.21	0.90	1.00	0.01	254.01	3.56
3	0.21	0.90	1.00	0.01	250.92	3.50
4	0.22	0.90	1.00	0.01	247.87	3.45
5	0.22	0.90	1.00	0.01	244.85	3.40
6	0.23	0.90	1.00	0.01	241.86	3.35

Table 2: Optimum settings for different compression ratios with predicted values of time, MSE and cost.

Testing Performance

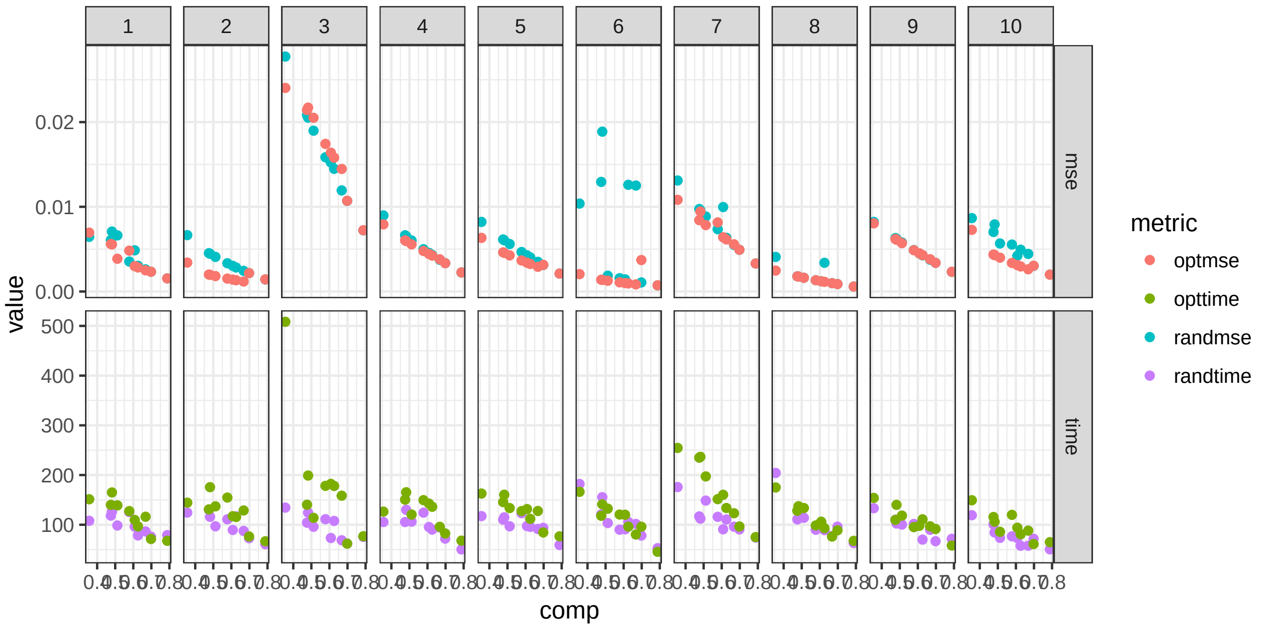
We now test whether or not the optimal settings perform as designed and calculate the amount of improvement acheived on average. To conduct such a test we get the same 10 images that were used before and randomly select 10 values of compression ratio to test on each image. Then we test random initialization at each of these settings to establish the baseline performance. Next, we perform the compression and decompression procedure for the same compression ratios but this time using the optimal settings we determined.

The testing is conducted and results are imported in R for analysis. These are shown using the code below.

```
# Compare the mse and time between optimal runs and random runs
simdes <- read.csv('simruns.csv')
randresp <- read.csv('randout.csv', header = FALSE) %>%
  rename(mse = V1, time = V2)
optresp <- read.csv('optout.csv', header = FALSE) %>%
  rename(mse = V1, time = V2)
simresults <- cbind(simdes[rep(c(1:10), 10),],randresp,optresp,
                    c(rep(1, 10), rep(2, 10), rep(3, 10), rep(4, 10), rep(5, 10),
                      rep(6, 10), rep(7, 10), rep(8, 10), rep(9, 10), rep(10, 10)))
```

```r
colnames(simresults) = c("comp","propedge","thick","msepred",
                         "timepred","cost","randmse","randtime",
                         "optmse","opttime","block")

# Plot the results
plotdata <- gather(simresults, metric, value, randmse, optmse, randtime, opttime) %>%
  cbind(type = c(rep('mse', 200), rep('time', 200)))
ggplot(aes(x = comp, y = value, color = metric), data = plotdata) +
  geom_point() +
  facet_grid(type~block, scales = "free_y")+
  theme_bw()
```



We can quantify these results by computing the average values of MSE and time by block and also overall.

```r
cost <- function(mse, time){
  f1 = 0
  f2 = 0
  f3 = 0
  if(mse>0.0125){
    f1 <- (mse+1)^100
  }
  if (time>240){
    f2 <- (1+(time/20000))^100
  }
  f3 <- mse + time/20000
  return(f1+f2+f3)
}
# Average values of MSE and time by block
group_by(simresults, block) %>%
  summarize(mean(randmse)-mean(optmse), mean(randtime)-mean(opttime))
```

```
## # A tibble: 10 x 3
##    block `mean(randmse) - mean(optmse)` `mean(randtime) - mean(opttime)`
##    <dbl>                          <dbl>                            <dbl>
## 1     1                    0.000515190                          -19.1410
## 2     2                    0.001673920                          -24.1073
## 3     3                   -0.000621440                          -83.9569
## 4     4                    0.000305090                          -26.3089
## 5     5                    0.000946490                          -26.1793
## 6     6                    0.005957867                           -3.5670
## 7     7                    0.000767250                          -53.1755
## 8     8                    0.000389225                           -2.8656
## 9     9                    0.000040690                          -12.6097
## 10    10                   0.001641740                          -20.0285
```

```
# Average values of MSE, time and cost
mutate(simresults, randcost = mapply(cost, simresults$randmse, simresults$randtime), optcost = mapply(co
  summarize(mean(randmse),mean(optmse), mean(randtime),mean(opttime), mean(randcost), mean(optcost))
```

```
##   mean(randmse) mean(optmse) mean(randtime) mean(opttime) mean(randcost)
## 1   0.006150403  0.004988801       99.28816      126.4821      0.7288362
##   mean(optcost)
## 1     0.7194613
```

These results show that overall the optimized procedure leads to a tradeoff such that there is a 20% increase in quality but a 27% increase in computational time.

Appendix: MATLAB Code

The MATLAB code is avaiable in the Github repository: https://github.com/arjunkakkar8/STAT-compression.