PHC 6068 Course Project

Tyler Grimes

December 12, 2016

1. Gelfand et al. (1990)

The joint distribution of Y, X, and all of the parameters can be factored based on the hierarchical structure.

$$\begin{aligned}
&P(Y, X, \alpha, \beta, \sigma_c^{-2}, \alpha_c, \sigma_c^{-2}, \beta_c, \sigma_\beta^{-2}) \\
&\quad = \prod_{ij} \left[P(Y_{ij} | X_j, \alpha_i, \beta_i, \sigma_c^{-2}) \right] \prod_i \left[P(\alpha_i | \alpha_c, \sigma_\alpha^{-2}) \right] \prod_i \left[P(\beta_i | \beta, \sigma_\beta^{-2}) \right] \\
&\qquad \times \; P(\alpha_c) P(\sigma_\alpha^{-2}) P(\beta_c) P(\sigma_\beta^{-2}) P(\sigma_c^{-2})
\end{aligned}$$

The conditional posterior distributions for each parameter are obtained from the joint distribution by,

$$\begin{aligned}
P(\alpha_i | Y, X, \beta, \sigma_c^{-2}, \alpha_c, \sigma_\alpha^{-2}, \beta_c, \sigma_\beta^{-2}) &\propto \prod_j \left[P(Y_{ij} | X_i, \alpha_i, \beta_i, \sigma_c^{-2}) \right] P(\alpha_i | \alpha_c, \sigma_\alpha^{-2}) \\
&\propto \exp\left(-\frac{1}{2\sigma_c^2} \sum_j (Y_{ij} - \alpha_i - \beta_i (x_j - \bar{x}))^2 \right) \exp\left(-\frac{1}{2\sigma_\alpha^2} (\alpha_i - \alpha_c)^2 \right) \\
&\propto \exp\left(-\frac{1}{2\sigma_c^2} \sum_j \left(\alpha_i^2 - 2\alpha_i (Y_{ij} - \beta_i (x_j - \bar{x})) \right) \right) \exp\left(-\frac{1}{2\sigma_\alpha^2} (\alpha_i^2 - 2\alpha_i \alpha_c \right) \\
&\propto \exp\left(-\frac{1}{2} \left[\alpha_i^2 \left(\frac{J}{\sigma_c^2} + \frac{1}{\sigma_\alpha^2} \right) - 2\alpha_i \left(\frac{\sum_j \left(Y_{ij} - \beta_i (x_j - \bar{X}) \right)}{\sigma_c^2} + \frac{\alpha_c}{\sigma_\alpha^2} \right) \right] \right) \\
&\propto \exp\left(-\frac{1}{2} \left[\alpha_i^2 \left(\frac{J}{\sigma_c^2} + \frac{1}{\sigma_\alpha^2} \right) - 2\alpha_i \left(\frac{\sum_j Y_{ij}}{\sigma_c^2} + \frac{\alpha_c}{\sigma_\alpha^2} \right) \right] \right) \\
&\propto \exp\left(-\frac{1}{2} A \left[\alpha_i - \frac{B}{A} \right]^2 \right) \\
&\sim \mathsf{N}\left(\frac{B}{A}, A^{-1/2} \right),
\end{aligned}$$

where $A = J/\sigma_C^2 + 1/\sigma_\alpha^2$ and $B = \alpha_c/\sigma_\alpha^2$. Similarly,

$$P(\beta_i | Y, X, \alpha, \sigma_c^{-2}, \alpha_c, \sigma_\alpha^{-2}, \beta_c, \sigma_\beta^{-2}) \propto \prod_j \left[P(Y_{ij} | X_i, \alpha_i, \beta_i, \sigma_c^{-2}) \right] P(\beta_i | \beta_c, \sigma_\beta^{-2})$$

$$
\begin{aligned}
&\propto \exp\left(-\frac{1}{2\sigma_c^2}\left[\sum_j\left(\beta_i^2(x_j-\bar{x})^2-2\beta_i(x_j-\bar{x})(Y_{ij}-\alpha_i)\right]-\frac{1}{2\sigma_\beta^2}(\beta_i^2-2\beta_i\beta_c\right)\right)\\
&\propto \exp\left(-\frac{1}{2}\left[\beta_i^2\left(\frac{\sum_j(x_j-\bar{x}}{\sigma_c^2}+\frac{1}{\sigma_\beta^2}\right)-2\beta_i\left(\frac{\sum_j(x_j-\bar{x})(Y_{ij}-\alpha_i)}{\sigma_c^2}+\frac{\beta_c}{\sigma_\beta^2}\right)\right]\right)\\
&\propto \exp\left(-\frac{1}{2}A\left[\alpha_i-\frac{B}{A}\right]^2\right)\\
&\sim \mathsf{N}\left(\frac{B}{A},A^{-1/2}\right),
\end{aligned}
$$

where $A=\sum_j(x_j-\bar{x})/\sigma_c^2+1/\sigma_\beta^2$ and $B=\sum_j(x_j-\bar{x})(Y_{ij}-\alpha_i)/\sigma_c^2+\beta_c/\sigma_\beta^2$.

$$
\begin{aligned}
P(\alpha_c|Y,X,\alpha,\beta,&\sigma_c^{-2},\sigma_\alpha^{-2},\beta_c,\sigma_\beta^{-2})\propto\prod_i\left[P(\alpha_i|\alpha_c,\sigma_\alpha^{-2})\right]P(\alpha_c)\\
&\propto \exp\left(-\frac{1}{2\sigma_\alpha^2}\sum_i(\alpha_i-\alpha_c)^2\right)\exp\left(-\frac{1}{2b^2}(\alpha_c-a)^2\right)\\
&\propto \exp\left(-\frac{1}{2}\left[\alpha_c^2\left(\frac{I}{\sigma_\alpha^2}+\frac{1}{b^2}\right)-2\alpha_c\left(\frac{\sum_i\alpha_i}{\sigma_\alpha^2}+\frac{a}{b^2}\right)\right]\right)\\
&\propto \exp\left(-\frac{1}{2}A\left[\alpha_i-\frac{B}{A}\right]^2\right)\\
&\sim \mathsf{N}\left(\frac{B}{A},A^{-1/2}\right),
\end{aligned}
$$

where $A=I/\sigma_\alpha^2+1/b^2$ and $B=\sum_i\alpha_i/\sigma_\alpha^2+a/b^2$. The distribution of $P(\alpha_c|...)$ is the same as what we just derived, but with a change of variables. That is,

$$
\begin{aligned}
P(\beta_c|Y,X,\alpha,\beta,&\sigma_c^{-2},\alpha_c,\sigma_c^{-2},\sigma_\beta^{-2})\propto\prod_i\left[P(\beta_i|\beta_c,\sigma_\beta^{-2})\right]P(\beta_c)\\
&\sim \mathsf{N}\left(\frac{B}{A},A^{-1/2}\right),
\end{aligned}
$$

where $A=I/\sigma_\beta^2+1/b^2$ and $B=\sum_i\alpha_i/\sigma_\beta^2+a/b^2$.

$$
\begin{aligned}
P(\sigma_c^{-2}|Y,X,\alpha,\beta,&\alpha_c,\sigma_\alpha^{-2},\beta_c,\sigma_\beta^{-2})\propto\prod_{ij}\left[P(Y_{ij}|x_i,\alpha_i,\beta_i,\sigma_c^{-2})\right]P(\sigma_c^{-2})\\
&\propto (\sigma_c^{-2})^{\frac{IJ}{2}}\exp\left(-\frac{1}{2\sigma_c^2}\sum_{ij}\left(Y_{ij}-\alpha_i-\beta_i(x_j-\bar{x})\right)^2\right)(\sigma_c^{-2})^{c-1}\exp\left(-\sigma_c^{-2}d\right)\\
&\propto (\sigma_c^{-2})^{\frac{IJ}{2}+c-1}\exp\left(-(\sigma_c^{-2})\left(\frac{1}{2}\sum_{ij}(Y_{ij}-\alpha_i-\beta_i(x_j-\bar{x})^2+d\right)\right)\\
&\sim \mathsf{Gamma}\left(\frac{IJ}{2}+c,A\right),
\end{aligned}
$$

where $A = \sum_{ij}(Y_{ij} - \alpha_i - \beta_i(x_j - \bar{x})^2/2 + d$.

$$\begin{aligned} P(\sigma_\alpha^{-2}|Y, X, \alpha, \beta, \sigma_c^{-2}, \alpha_c, \beta_c, \sigma_\beta^{-2}) &\propto \prod_i \left[P(\alpha_i|\alpha_c, \sigma_\alpha^{-2})\right] P(\sigma_\alpha^{-2}) \\ &\propto (\sigma_\alpha^{-2})^{\frac{I}{2}} \exp\left(-\frac{1}{2\sigma_\alpha^2}\sum_{ij}(\alpha_i - \alpha_c)^2\right)(\sigma_\alpha^{-2})^{c-1}\exp\left(-\sigma_\alpha^{-2}d\right) \\ &\propto (\sigma_\alpha^2)^{\frac{I}{2}+c-1}\exp\left(-(\sigma_\alpha^{-2})\left(\frac{1}{2}\sum_i(\alpha_i - \alpha_c)^2 + d\right)\right) \\ &\sim \mathsf{Gamma}\left(\frac{IJ}{2} + c, A\right), \end{aligned}$$

where $A = \sum_i(\alpha_i - \alpha_c)^2/2 + d$. The conditional distribution of $P(\sigma_\beta^{-2}|...)$ is the same as the previous derivation with a change of variables.

$$\begin{aligned} P(\sigma_\beta^{-2}|Y, X, \alpha, \beta, \sigma_c^{-2}, \alpha_c, \sigma_\alpha^{-2}, \beta_c) &\propto \prod_i \left[P(\beta_i|\alpha_c, \sigma_\beta^{-2})\right] P(\sigma_\beta^{-2}) \\ &\sim \mathsf{Gamma}\left(\frac{IJ}{2} + c, A\right), \end{aligned}$$

where $A = \sum_i(\beta_i - \beta_c)^2/2 + d$.

These distributions can be used to perform Gibbs sampling and obtain estimates for the model parameters.

Part A.

The rats data set is loaded and passed into Gibbs. The sampler is initially run for 10000 iterations and convergence is checked. 1 shows the chain of samples for each α_i, β_i, and σ_c. It appears that each chain converges within the first 100 iterations. The sampler is continued for another 1000 iterations; the average of these values is used for the Gibbs estimates $\hat{\alpha}_i$, $\hat{\beta}_i$, and $\hat{\sigma}_c$. These estimates are shown in 2, and the resulting regression lines are shown with respect to the rats data set in 4.

Part B.

To assess the performance of the Gibbs estimators, data sets are simulated which reflect the rats data set. In particular, the α_i are generated from a $\mathsf{N}(240, 14)$, the β_i from a $\mathsf{N}(6, 0.5)$, and σ_c from a $\mathsf{N}(6, 1.2)$. The parameter values for these distributions were chosen so that the simulated data sets will be similar in scale to the rats data. The relative errors for each estimate are computed and their distributions are shown in 4. Overall, the Gibbs estimates are within 20% of the true parameter value; the largest variation is found in the estimator for σ_c. We are most interested in the estimates for the slopes, $\hat{\beta}_i$, and these tend to be within 10% of the true value.

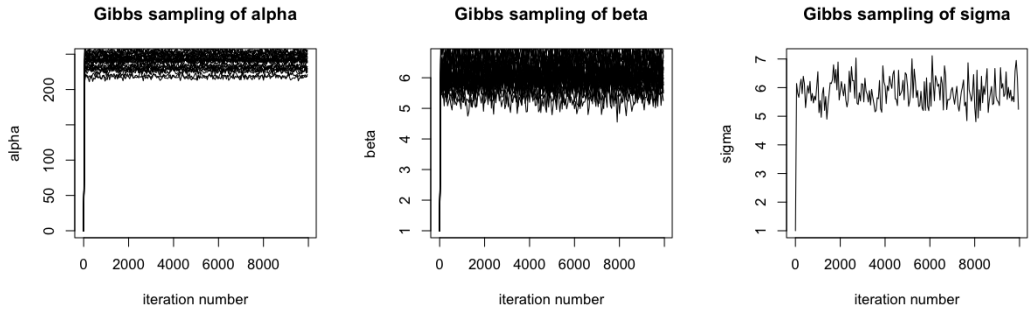

Figure 1: Distribution of Gibbs samples for each α_i, β_i and σ_c over the first 10000 iterations. The samples appear to converge within the first 100 samples.

```
$alpha_hat
 [1] 239.8397 247.8114 252.3103 232.4013 231.4923 249.6780 228.6384
 [8] 248.3310 281.4071 219.1859 258.2680 228.0564 242.3163 268.1319
[15] 242.6524 245.3007 232.1269 240.3658 253.8781 241.3963 248.5920
[22] 225.1007 228.4898 245.1309 234.4164 253.8918 254.4517 242.8945
[29] 217.8928 241.4593

$beta_hat
 [1] 6.063419 7.045638 6.476143 5.332463 6.561886 6.178999 5.970513
 [8] 6.426021 6.942604 5.839820 6.817056 6.120210 6.159408 6.692181
[15] 5.413839 5.915959 6.293556 5.847658 6.394368 6.041722 6.399854
[22] 5.862524 5.745468 5.878479 6.916064 6.552598 5.904618 5.835923
[29] 5.679890 6.121373

$sigma_c_hat
[1] 5.907456
```

Figure 2: Estimates obtained from the average of 1000 Gibbs samples.

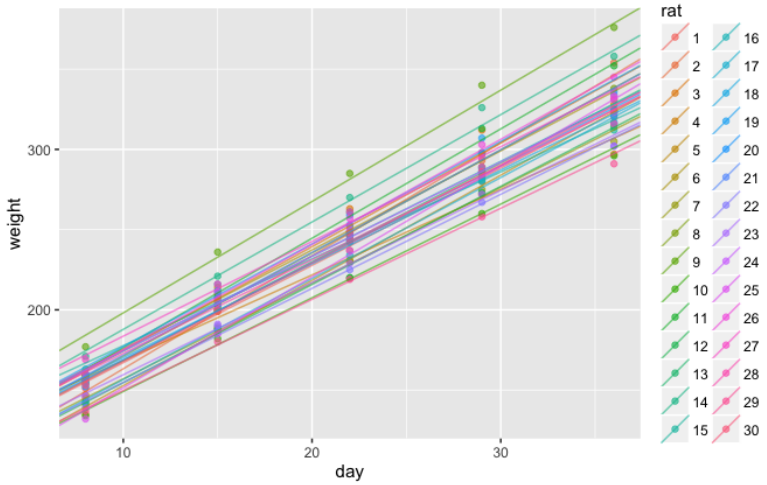

Figure 3: Regression lines from Gibbs estimates for each of the 30 rats.

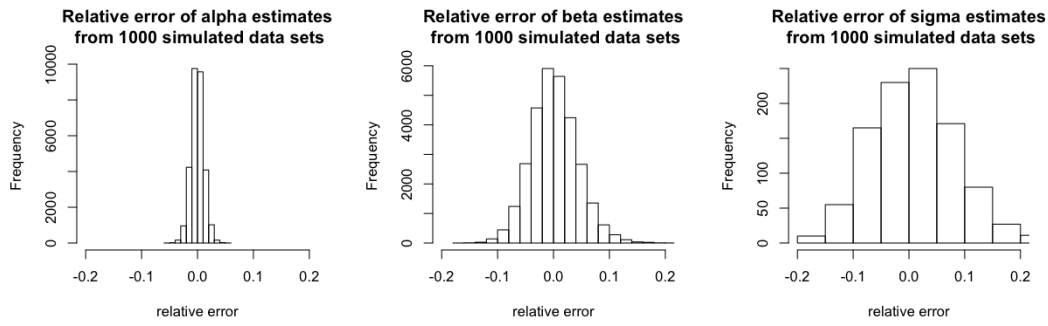

Figure 4: Distribution of relative errors for $\hat{\alpha}_i$, $\hat{\beta}_i$, $\hat{\sigma}_c$ from 1000 generated data sets. The data sets are generated based on the Bayesian model with parameter values chosen to mimic the rats dataset.

R code for question 1

```
#Gelfand and Smith 1990, JASA
set.seed(10000)

I <- 30 #i = 1, ..., I   Number of rats.
J <- 5  #j = 1, ..., J   Number of measurements.

X <- c(8, 15, 22, 29, 36)
xbar <- mean(X)

#Hyperparameters.
a <- 0       #alpha_c ~ N(a, b)
b <- sqrt(1000)     #alpha_c ~ N(a, b)
c <- 0.01    #sigma_a,b,c^(-2) ~ Gamma(c, d)
d <- 0.01

#Conditional distributions.
# Y : I by J matrix containing the data set.
# X : J by 1 vector of x values.
# alpha : I by 1 vector of alpha values.
# beta : I by 1 vector of beta values.
# alpha_c, beta_c, sigma_a, sigma_b, sigma_c : all scalars.
cond_alpha <- function(Y, alpha_c, sigma_a_n2, sigma_c_n2) {
  A <- J * sigma_c_n2 + sigma_a_n2
  B <- (Y %*% rep(1, J)) * sigma_c_n2 + alpha_c * sigma_a_n2
  rnorm(I, B/A, A^(-0.5))
}
cond_beta<- function(Y, alpha, beta_c, sigma_b_n2, sigma_c_n2) {
  A <- sum((X - xbar)^2) * sigma_c_n2 + sigma_b_n2
  B <- (Y %*% (X - xbar)) * sigma_c_n2 + beta_c * sigma_b_n2
  rnorm(I, B/A, A^(-0.5))
```

```
}
cond_alpha_c <- function(alpha, sigma_a_n2) {
  A <- I * sigma_a_n2 + 1/b^2
  B <- sum(alpha) * sigma_a_n2 + a / b^2
  rnorm(1, B/A, A^(-0.5))
}
cond_beta_c <- function(beta, sigma_b_n2) {
  A <- I * sigma_b_n2 + 1/b^2
  B <- sum(beta) * sigma_b_n2 + a / b^2
  rnorm(1, B/A, A^(-0.5))
}
cond_sigma_a_n2 <- function(alpha, alpha_c) {
  A <- 0.5 * sum((alpha - alpha_c)^2) + d
  rgamma(1, I/2 + c, rate = A)
}
cond_sigma_b_n2 <- function(beta, beta_c) {
  A <- 0.5 * sum((beta - beta_c)^2) + d
  rgamma(1, I/2 + c, rate = A)
}
cond_sigma_c_n2 <- function(Y, alpha, beta) {
  Z <- Y - alpha - (beta %*% t(X - xbar))
  A <- 0.5 * sum(Z^2) + d
  rgamma(1, I*J/2 + c, rate = A)
}


#Gibbs sampler.
Gibbs <- function(Y, BURN = 100, ITER = 1000, init_alpha = rep(0, I),
                  init_beta = rep(1, I), init_alpha_c = 1,
                  init_beta_c = 1, init_sigma_a = 1,
                  init_sigma_b = 1, init_sigma_c = 1,
                  GRAPH_DIAGNOSTIC = TRUE) {

  #Store estimates of alpha, beta, and csigma.
  alpha <- matrix(0, nrow = BURN + ITER, ncol = I)
  beta <- matrix(0, nrow = BURN + ITER, ncol = I)
  sigma_c_n2 <- rep(0, I)

  #Initialize estimates.
  alpha[1, ] <- init_alpha
  beta[1, ] <- init_beta
  sigma_c_n2[1] <- init_sigma_c^(-2)
  alpha_c <- init_alpha_c
  beta_c <- init_beta_c
  sigma_a_n2 <- init_sigma_a^(-2)
  sigma_b_n2 <- init_sigma_b^(-2)
```

```
  for(i in 2:(BURN + ITER)) {
    alpha[i, ] <- cond_alpha(Y, alpha_c, sigma_a_n2, sigma_c_n2[i - 1])
    beta[i, ] <- cond_beta(Y, alpha[i, ], beta_c,
                           sigma_b_n2, sigma_c_n2[i - 1])
    sigma_c_n2[i] <- cond_sigma_c_n2(Y, alpha[i, ], beta[i, ])
    sigma_a_n2 <- cond_sigma_a_n2(alpha[i, ], alpha_c)
    sigma_b_n2 <- cond_sigma_b_n2(beta[i, ], beta_c)
    alpha_c <- cond_alpha_c(alpha[i, ], sigma_a_n2)
    beta_c <- cond_beta_c(beta[i, ], sigma_b_n2)
  }

  if(GRAPH_DIAGNOSTIC) {
    iterations <- seq(1, BURN, ceiling(0.005*BURN))
    png("images/alpha_diagnostic.png", 400, 400, res = 100)
    plot(x = iterations, y = alpha[iterations, 1], type = "l",
         main = "Gibbs sampling of alpha",
         xlab = "iteration number", ylab = "alpha")
    for(i in 1:I) {
      lines(x = iterations, y = alpha[iterations, i])
    }
    dev.off()

    png("images/beta_diagnostic.png", 400, 400, res = 100)
    plot(x = iterations, y = beta[iterations, 1], type = "l",
         main = "Gibbs sampling of beta",
         xlab = "iteration number", ylab = "beta")
    for(i in 1:I) {
      lines(x = iterations, y = beta[iterations, i])
    }
    dev.off()

    png("images/sigma_diagnostic.png", 400, 400, res = 100)
    plot(x = iterations, y = sigma_c_n2[iterations]^(-1/2),
         type = "l", main = "Gibbs sampling of sigma",
         xlab = "iteration number", ylab = "sigma")
    dev.off()
  }
  alpha_hat <- apply(alpha[(BURN + 1):(BURN + ITER), ], 2, mean)
  beta_hat <- apply(beta[(BURN + 1):(BURN + ITER), ], 2, mean)
  sigma_c_hat <- mean(sigma_c_n2[(BURN + 1):(BURN + ITER)]^(-1/2))

  return(list(alpha_hat = alpha_hat,
              beta_hat = beta_hat,
              sigma_c_hat = sigma_c_hat))
}

# ##################################################
```

```
#
# Part A.
#
# ######################################################
library(DPpackage)
data(rats)
rats$rat <- as.factor(rats$rat)
Y <- matrix(rats$weight, nrow = 30, ncol = 5, byrow = TRUE)

#Run Gibbs.
results <- Gibbs(Y, BURN = 10000)
results

#Graph the resulting regression lines onto the rats data.
Gibbsframe <- data.frame(rat = as.factor(1:30),
                         alpha_hat = results$alpha_hat,
                         beta_hat = results$beta_hat)

png("images/gibbs_regression.png", 680, 420, res = 100)
ggplot() +
  geom_point(data = rats, aes(day, weight, color = rat),
             alpha = 0.6) +
  geom_abline(data = Gibbsframe, alpha = 0.6,
              aes(intercept = alpha_hat - beta_hat*xbar, slope = beta_hat, co
dev.off()

# ######################################################
#
# Part B.
#
# ######################################################
#Generate a data set from specified parameter values.
#Note, sigma_a, sigma_b, and sigma_c are requested, not sigma^(-2).
generate_data <- function(alpha_c = 0, beta_c = 0, sigma_a = 1,
                          sigma_b = 1, sigma_c = 1) {
  #Generate alphas and betas.
  alpha <- rnorm(I, alpha_c, sigma_a)
  beta <- rnorm(I, beta_c, sigma_b)
  sigma_c <- abs(rnorm(1, sigma_c, 0.2*sigma_c))

  #Generate y's.
  Y <- matrix(rnorm(I*J, alpha + beta %*% t(X - xbar),
                    sigma_c), nrow = I, ncol = J)

  return(list(Y = Y,
              params = list(alpha = alpha, beta = beta,
                            sigma_c= sigma_c)))
```

```
}

#Compare the Gibbs estimates to the true parameters and
# summarize the results.
summarize <- function(results, params, GRAPH = TRUE) {
  #Analyze the results.
  true <- c(params$alpha, params$beta, params$sigma_c)
  pred <- c(results$alpha_hat, results$beta_hat,
            results$sigma_c_hat)
  label <- c(rep("alpha", I), rep("beta", I), "sigma")

  #Make scatterplot of relative errors.
  g <- NULL
  if(GRAPH) {
    g <- ggplot(data.frame(residuals = (pred - true)/true,
                           rat = c(rep(1:I, 2), I/2), label = label),
                aes(rat, residuals), environment = environment()) +
      geom_point(aes(color = label), alpha = 0.7) +
      geom_abline(slope = 0, intercept = 0) +
      ylab("relative error")

    g
  }

  residuals <- (pred - true)/true
  alpha_residuals <- residuals[1:I]
  beta_residuals <- residuals[(I + 1):(2*I)]
  sigma_residual <- residuals[2*I + 1]

  return(list(alpha_residuals = alpha_residuals,
              beta_residuals = beta_residuals,
              sigma_residual = sigma_residual,
              g = g))
}

#Perform simulations using parameter values reflect
# the rats data set.
sim <- 1000
alpha_residuals <- rep(0, I*sim)
beta_residuals <- rep(0, I*sim)
sigma_residuals <- rep(0, sim)
for(i in 1:sim) {
  data <- generate_data(240, 6, 14, 0.5, 6)
  results <- Gibbs(data$Y, ITER = 10000,
                   GRAPH_DIAGNOSTIC = FALSE)
  s <- summarize(results, data$params, GRAPH = FALSE)
  alpha_residuals[(1 + I*(i - 1)):(I*i)] <- s$alpha_residuals
```

```
  beta_residuals[(1 + I*(i - 1)):(I*i)] <- s$beta_residuals
  sigma_residuals[i] <- s$sigma_residual
  #ggsave(paste("images/results_rats_sim_", i, ".png", sep = ""))
}

png("images/alpha_rel_errors.png", 400, 400, res = 100)
hist(alpha_residuals, xlim = c(-0.2, 0.2),
     xlab = c("relative error"),
     main = paste("Relative error of alpha estimates\n",
                  "from", sim, "simulated data sets"))
dev.off()

png("images/beta_rel_errors.png", 400, 400, res = 100)
hist(beta_residuals, xlim = c(-0.2, 0.2),
     xlab = c("relative error"),
     main = paste("Relative error of beta estimates\n",
                  "from", sim, "simulated data sets"))
dev.off()

png("images/sigma_rel_errors.png", 400, 400, res = 100)
hist(sigma_residuals, xlim = c(-0.2, 0.2),
     xlab = c("relative error"),
     main = paste("Relative error of sigma estimates\n",
                  "from", sim, "simulated data sets"))
dev.off()
```

2. Yu et al. (2011)

Implement the stochastic approximation Monte Carlo (SAMC) algorithm for evaluating p-values of the two-sample t-test using a simulated dataset.

Results

The randomization procedure and SAMC algorithm were both implemented in R. For SAMC, the sample space was partitioned into $m = 101$ subsets with the lower and upper thresholds set at 0 and 7, respectively. π is set to equally weighted values $\pi_i = 1/m$, and the gain factor is $\gamma_i = t_0/max(t_0, i)$ with $t_0 = 5000$. The p-value for a t-statistic λ in the subinterval $(\lambda_{k-1}, \lambda_k]$ is calculated by interpolating

$$\hat{F}(\lambda) = \frac{\lambda - \lambda_{k-1}}{\lambda_k - \lambda_{k-1}}\hat{F}(\lambda_k) + \frac{\lambda_k - \lambda}{\lambda_k - \lambda_{k-1}}\hat{F}(\lambda_{k-1})$$

where $\hat{F}(\lambda_k) = \frac{\sum_{i=1}^{k} \exp(\hat{\theta}_i)\pi_i}{\sum_{i=1}^{m} \exp(\hat{\theta}_i)\pi_i}$ and the p-value given by $1 - \hat{F}(\lambda)$; i.e., we consider the upper-tail probabilities.

A dataset was simulated with $Y_1 \sim N_{n_1}(\mu_1, \sigma_1)$ and $Y_2 \sim N_{n_2}(\mu_2, \sigma_2)$ with $\mu_1 = \mu_2 = 0$, $\sigma_1 = \sigma_2 = 1$ and $n_1 = n_2 = 1000$. In each iteration of the randomization and SAMC procedures, a permuted dataset was obtained by randomly selecting 5% of of the observations to swap between Y_1 and Y_2. Both procedures were ran for 10^6 iterations.

The sampling distribution obtained by the randomization procedure is shown in 5. This procedure gives a good approximation to the true distribution, but fails to produce estimates for extreme t-statistics. To reach these extreme values, we need to run on the order of 10^8 or higher iterations. The SAMC procedure does not have this issue, and can produce estimates for extreme values with only 10^6 iterations. The estimates for such p-values are shown in 6 with their absolute relative error ARE $= |\hat{p} - p|/p$. It appears that with the chosen parameters for E, m, and $t0$, the resulting estimates tend to underestimate the true p-values.

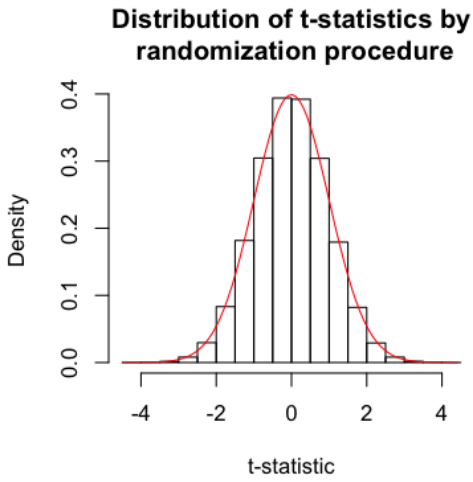

Figure 5: Estimated distribution of t-statistics from randomization procedure after 10^6 iterations. The actual distribution, t_{1998}, is shown in red.

R code for question 2

```
set.seed(1000)
library(gridExtra)

#Generate data set for t-test.
generate_data <- function(n = 1000, mu = c(0, 0), sigma = c(1, 1)) {
  Y <- cbind(rnorm(n, mu[1], sigma[1]),
             rnorm(n, mu[2], sigma[2]))
  colnames(Y) <- c("X", "Y")

  return(Y)
}

#Calculate t-test statistic for a given data set Y.
```

True P-value	SAMC	ARE%
0.01	0.00912	8.8
1e-03	8.93e-04	10.7
1e-04	8.28e-05	17.2
1e-05	6.69e-06	33.1
1e-06	5.96e-07	40.4
1e-07	4.09e-08	59.1
1e-08	3.43e-09	65.7
1e-09	2.91e-10	70.9
1e-10	2.01e-11	79.9

Figure 6: The estimates and ARE% are shown for various extreme p-values by the SAMC procedures after 10^6 iterations. The randomization procedure typically needs 10^8 or 10^9 iterations before producing samples within these extreme ranges, and so it's estimates for these p-values are 0; those results are ommited.

```
t_val <- function(Y) {
  if(any(is.na(Y))) {
    error("Y contains NA values.")
  }
  t <- as.numeric(t.test(Y[, 2], Y[, 1], var.equal = TRUE)$statistic)
}

# ######################################################
#
# Randomization test
#
# ######################################################
#Returns the right tail probability at t.
randomization_p_value<-function(t, t_sample){
  pval <- sum(t_sample[t_sample >= t])/length(t_sample)

  return(pval)
}

randomization <- function(Y, MAX_ITER = 10^5,
                          PERMUTATION_PERCENT = 0.05) {
  n <- nrow(Y)
  #Number of observations to swap:
```

```
  swap_size <- ceiling(n*PERMUTATION_PERCENT)
  t_sample <- rep(0, MAX_ITER)
  t_sample[1] <- t_val(Y)

  for(i in 1:MAX_ITER) {
    swap_indicies <- sample(1:n, swap_size)
    Y_new <- Y
    Y_new[swap_indicies, ] <- Y_new[swap_indicies, c(2, 1)]
    t_sample[i] <- t_val(Y_new)
    Y <- Y_new
  }

  return(t_sample)
}

# ######################################################
#
# SAMC
#
# ######################################################
#Returns the right tail probability P(T >= t)
SAMC_p_value<-function(t, E, theta, PI){
  m <- length(theta)
  psi <- exp(theta)*PI #Assuming no empty subregions.
  normalize <- sum(psi) #Normalizing factor.
  p <- psi/normalize
  pval <- 0

  index <- get_partition_index(t, E)
  if(index == 1) {
    pval <- 1
  } else {
    pval <- (t - E[index - 1])/
      (E[index] - E[index - 1])*sum(p[1:index])
    pval <- pval + (E[index] - t)/
      (E[index] - E[index - 1])*sum(p[1:(index - 1)])
    pval <- 1 - pval
  }

  return(pval)
}

#Find which subset t belongs to in the sampling space partition.
get_partition_index<-function(t, E){
  m <- length(E)
  #First check edge cases.
  if(t < E[1]) {
```

```
    return(1)
  } else if (t >= E[m]) {
    return(m + 1)
  }

  #Binary search to find which subset contains t.
  binary_search <- function(t, indicies) {
    m <- length(indicies)
    if(m == 1) {
      if(t < E[indicies]) {
        return(indicies)
      } else {
        return(indicies + 1)
      }
    }
    i <- ceiling(m/2) + 1 #Start in the middle of the set of indicies.
    if(t >= E[indicies[i - 1]] && t < E[indicies[i]]) {
      return(indicies[i])
    }
    if(t < E[indicies[i - 1]]) {
      return(binary_search(t, indicies[1:(i - 1)]))
    } else {
      return(binary_search(t, indicies[i:m]))
    }
  }

  return(binary_search(t, 2:m))
}

SAMC <- function(Y, MAX_ITER = 10^5, E = seq(0, 7, length.out = 100),
                 PI = NULL, t0 = 5000, PERMUTATION_PERCENT = 0.05) {
  n <- nrow(Y)
  m <- length(E)
  #Number of observations to swap:
  swap_size <- ceiling(n*PERMUTATION_PERCENT)

  #Initialize variables for SAMC.
  PI <- rep(1/(m + 1), m + 1)
  gamma <- t0/(pmax(t0, 1:MAX_ITER))
  theta <- rep(0, m + 1)

  #Find the sampling partition index of the current t-statistic.
  t <- t_val(Y)
  partition_index <- get_partition_index(t, E)

  for(i in 2:MAX_ITER) {
    #Generate a permutation from the given sample.
```

```
    swap_indicies <- sample(1:n, swap_size)
    Y_new <- Y
    Y_new[swap_indicies, ] <- Y_new[swap_indicies, c(2, 1)]

    #Find the partition index of this t-statistic.
    t <- t_val(Y_new)
    partition_index_new <- get_partition_index(t, E)

    #Compute the MH ratio.
    r <- exp(theta[partition_index] - theta[partition_index_new])

    if(r > runif(1)) {
      Y <- Y_new
      partition_index <- partition_index_new
    } else {
      #Do nothing.
    }

    #Update theta (based on whether this t-statistic was accepted).
    indicator <- rep(0, m + 1)
    indicator[partition_index] <- 1
    theta <- theta + gamma[i]*(indicator - PI)
  }

  return(list(theta = theta, PI = PI, E = E))
}

# ######################################################
#
# Results
#
# ######################################################
p <- 1*10^(-(2:10)) #pvalues to consider
randomization_iter <- 10^6
SAMC_iter <- 10^6
estimates <- matrix(0, nrow = length(p), ncol = 5)
colnames(estimates) <- c("True p-value", "Randomization", "SAMC",
                         "ARE% (Rand)", "ARE% (SAMC)")
#Repeat simulation for 20 different data sets generated under H0.
Y <- generate_data()
n <- nrow(Y)

#Randomization estimate from sample.
t_sample <- randomization(Y, randomization_iter)

#SAMC estimate from sample.
results <- SAMC(Y, SAMC_iter)
```

```r
#Compute p-values for different t-statistic values
for(j in 1:length(p)) {
  #Obtain t statistic for desired p-value.
  t <- qt(1 - p[j], 1998)

  #Randomization estimate.
  randomization_val <- randomization_p_value(t, t_sample)
  randomization_ARE <- abs((randomization_val - p[j])/p[j])*100

  #SAMC estimates:
  SAMC_val <- SAMC_p_value(t, results$E, results$theta, results$PI)
  SAMC_ARE <- abs((SAMC_val - p[j])/p[j])*100

  estimates[j, ] <-
    c(p[j], randomization_val, SAMC_val, randomization_ARE, SAMC_ARE)
}

png("images/p_val_rand_dist.png", 400, 400, res = 100) {
  hist(t_sample, main = "Distribution of t-statistics\n
       by randomization procedure",
       xlab = "t-statistic", freq = FALSE)
  curve(pt(x, 1998), min(t_sample), max(t_sample))
}

tab1 <- estimates[, c(1, 3, 5)]
tab1[, 3] <- round(tab1[, 3], 1)
tab1[, 1] <- signif(tab1[, 1], digits = 1)
tab1[, 2] <- signif(tab1[, 2], digits = 3)
colnames(tab1) <- c("True P-value", "SAMC", "ARE%")
png("images/p_val_ARE.png", 400, 400, res = 100)
grid.table(tab1, rep("", nrow(tab1)))
dev.off()
```

4. Liang et al. (2014)

Given a set of cities, the traveling salesman problem (TSP) is to find the shortest tour which goes through each city once and only once. Implement the simulated annealing and simulated stochastic approximation annealing algorithms for a traveling salesman problem for which 100 cities are uniformly distributed on a square region of length 100 miles. Compare the performance of the two algorithms and find the average length of the shortest tour (over 100 datasets).

Results

The SA and SAA algorithms were both implemented in R. For both algorithms, the initial temperature was set to $t_0 = 10$ with a square-root cooling schedule; the temperature at the i^{th} iteration is $t_i = t_0/\sqrt{(i)}$. Each algorithm is run for 10^5 iterations.

The results for one simulated data set are shown in 7 and 8. These images show the tour at different iteration steps to illustrate how the tour evolves. For the first dataset, SAA found a shorter tour which is about 60 miles shorter than the optimal found by SA. It also appears to converge to an optimum much earlier than SA. One hundred total city configurations were generated; the length of the shortest tours were recorded and are displayed in the scatterplot in 9. Interestingly, the optimal tour lengths found by each algorithm do not seem to be correlated. That is, if one algorithm performs poorly for a given arrangement of cities, the other algorithm may perform better.

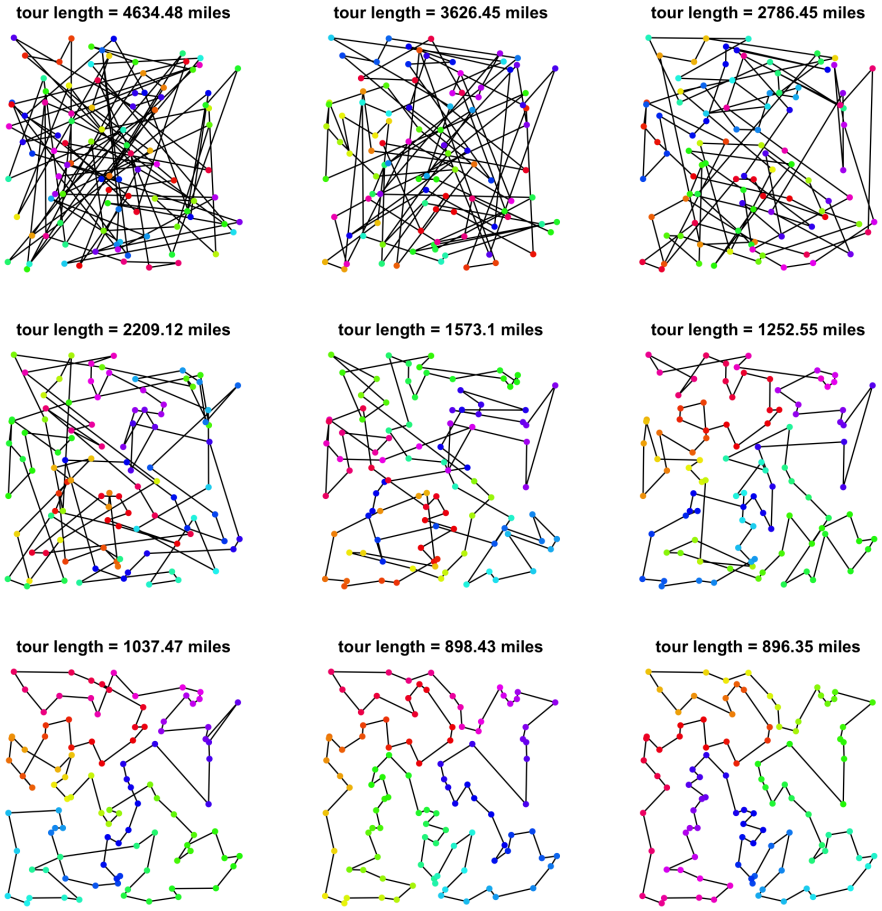

Figure 7: Tours found by SA at different iteration steps.

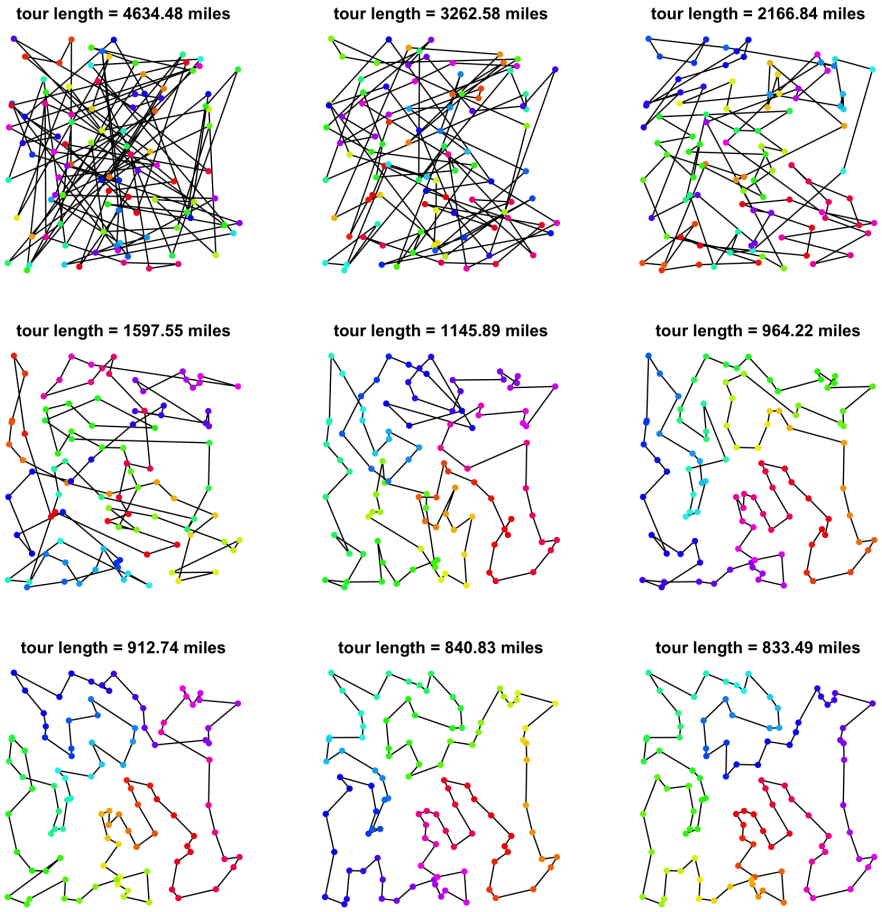

Figure 8: Tours found by SAA at different iteration steps.

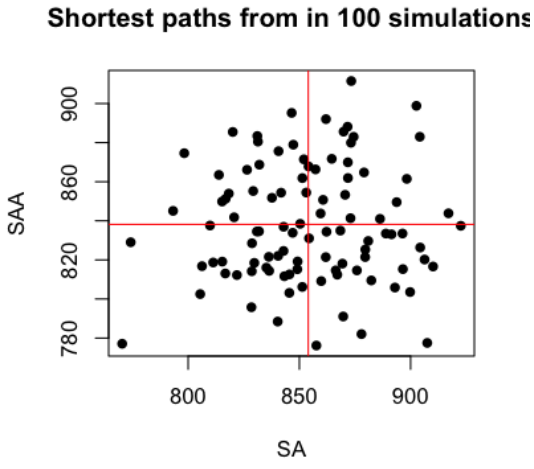

Figure 9: Optimal tour lengths of SA against SAA for 100 different city arrangements. The average length for SA and SAA are 854.0 miles and 838.2 miles, respectively.

R code for question 4

```
set.seed(10)

# ######################################################
#
# Auxillary functions.
#
# ######################################################
#Generate n cities from a height by width square mile region.
generate_cities <- function(n = 100, height = 100, width = 100) {
  cities <- matrix(0, nrow = n, ncol = 2)
  cities[, 1] <- runif(n, 0, width)
  cities[, 2] <- runif(n, 0, height)

  return(cities)
}

#Swap two elements in a vector.
swap <- function(x, move) {
  move <- sort(move)
  x[seq(move[1], move[2])] <- x[seq(move[2], move[1])]
  x
}

#Distance measure between two cities.
distance <- function(city_a, city_b) {
  d <- city_b - city_a
  dist <- t(d) %*% d
  sqrt(dist)
}

#Create a scatterplot representing a tour.
tour_plot <- function(tour, cities, POINTS = TRUE, ...) {
  len <- round(tour_length(tour, cities), digits = 2)
  plot(c(cities[tour, 1], cities[tour[1], 1]),
       c(cities[tour, 2], cities[tour[1], 2]),
       type = "l", main = paste("tour length =", len, "miles"),
       xlab = "", ylab = "", ...)
  if(POINTS) {
    color <- hsv(seq(0, 1, 1/n), 1, 0.95)
    points(cities[tour, 1], cities[tour, 2], pch = 16, col = color)
  }
}

#Compute the length of a tour.
tour_length <- function(tour, cities) {
```

```
  #Number of cities
  n <- length(tour)

  #Compute length of tour by summing the distances between each city.
  d <- 0
  for(i in 1:(n-1)) {
    #Distance between cities i and i+1.
    d <- d + distance(cities[tour[i], ], cities[tour[i+1], ])
  }
  #Loop end of tour back to beginning.
  d <- d + distance(cities[tour[n], ], cities[tour[1], ])

  d
}

#Compute the change in length of a tour after swapping cities i and j.
tour_change <- function(tour, cities, move) {
  #return(tour_length(swap(tour, move), cities) -
  #         tour_length(tour, cities))
  n <- nrow(cities)

  i <- move[1]
  j <- move[2]
  im1 <- ifelse(i - 1 < 1, n, i - 1)
  jp1 <- ifelse(j + 1 > n, 1, j + 1)

  new_distance <- distance(cities[tour[im1], ], cities[tour[j], ]) +
    distance(cities[tour[i], ], cities[tour[jp1], ])

  old_distance <- distance(cities[tour[im1], ], cities[tour[i], ]) +
    distance(cities[tour[j], ], cities[tour[jp1], ])

  new_distance - old_distance
}



# ######################################################
#
# Simulated annealing.
#
# ######################################################
#Simulated Annealing algorithm.
SA <- function(cities, tour = NULL, initial_t = 10, ITER = 10^5,
               HISTORY = TRUE) {
  n <- nrow(cities) #Number of cities
```

```
#Initialize a tour.
if(is.null(tour)) {
  tour <- sample(1:n, n)
}

#Initialize the temperature
t <- initial_t

#Keep track of all the tours (for plotting purposes later).
if(HISTORY) {
  tour_history <- matrix(0, nrow = ITER/2, ncol = n)
  tour_history[1, ] <- tour
  changes <- 2
}

#Begin SA:
for(iter in 1:ITER) {
  #Sample a new tour.
  move <- sort(sample(1:n, 2)) #Lin move. Choose two points to swap
  new_tour <- swap(tour, move)

  #Compute the change in distance of the tour length.
  delta_dist <- tour_change(tour, cities, move)

  #Determine if new tour should be accepted.
  if(delta_dist < 0) {
    tour <- new_tour
    if(HISTORY) {
      tour_history[changes, ] <- tour
      changes <- changes + 1
    }
  } else {
    u <- runif(1, 0, 1)
    if(exp(-delta_dist/t) > u) {
      tour <- new_tour
      if(HISTORY) {
        tour_history[changes, ] <- tour
        changes <- changes + 1
      }
    }
  }

  #Cool the temperature (square root cooling rate).
  t <- initial_t/sqrt(iter)
}

if(HISTORY) {
```

```
    tour_history = tour_history[1:(changes-1), ]
  } else {
    tour_history = NULL
  }

  return(list(shortest_tour = tour_length(tour, cities),
              tour_history = tour_history))
}


# ######################################################
#
# Stochastic approximation annealing.
#
# ######################################################
#Find which subset t belongs to in the sampling space partition.
get_E_index<-function(t, E){
  m <- length(E)
  #First check edge cases.
  if(t < E[1]) {
    return(1)
  } else if (t >= E[m]) {
    return(m + 1)
  }

  #Binary search to find which subset contains t.
  binary_search <- function(t, indicies) {
    m <- length(indicies)
    if(m == 1) {
      if(t < E[indicies]) {
        return(indicies)
      } else {
        return(indicies + 1)
      }
    }
    i <- ceiling(m/2) + 1 #Start in the middle of the set of indicies.
    if(t >= E[indicies[i - 1]] && t < E[indicies[i]]) {
      return(indicies[i])
    }
    if(t < E[indicies[i - 1]]) {
      return(binary_search(t, indicies[1:(i - 1)]))
    } else {
      return(binary_search(t, indicies[i:m]))
    }
  }

  return(binary_search(t, 2:m))
```

```
}

#Simulated Annealing algorithm.
SAA <- function(cities, tour = NULL, initial_t = 10, ITER = 10^5, HISTORY = T

  n <- nrow(cities) #Number of cities

  # Define initial values for SAMC
  E <- seq(750, 1000, length.out = 100)
  m <- length(E)
  theta <- rep(0, m + 1)
  PI <- 1/(m + 1) #(m + 1):1/(sum(1:(m + 1)))
  t0 <- 1000
  gamma <- t0/pmax(t0, 1:ITER)

  #Initialize a tour.
  if(is.null(tour)) {
    tour <- sample(1:n, n)
  }

  #Compute initial tour length.
  dist <- tour_length(tour, cities)

  #Find E index of current tour length.
  index <- get_E_index(dist, E)

  #Initialize the temperature
  t <- initial_t

  #Keep track of all the tours (for plotting purposes later).
  if(HISTORY) {
    tour_history <- matrix(0, nrow = ITER/2, ncol = n)
    tour_history[1, ] <- tour
    changes <- 2
  }

  #Begin SAA:
  for(iter in 1:ITER) {
    #Sample a new tour by a Lin move.
    move <- sort(sample(1:n, 2))
    new_tour <- swap(tour, move)

    #Compute the change in distance of the tour length.
    delta_dist <- tour_change(tour, cities, move)

    #Determine if new tour should be accepted.
    if(delta_dist < 0) {
```

```
      tour <- new_tour
      dist <- dist + delta_dist
      index <- get_E_index(dist, E)
      if(HISTORY) {
        tour_history[changes, ] <- tour
        changes <- changes + 1
      }
    } else {
      u <- runif(1, 0, 1)
      new_index <- get_E_index(dist + delta_dist, E)
      if(exp(-delta_dist/t + theta[index] - theta[new_index]) > u) {
        tour <- new_tour
        dist <- dist + delta_dist
        index <- new_index
        if(HISTORY) {
          tour_history[changes, ] <- tour
          changes <- changes + 1
        }
      }
    }

    #Update theta.
    indicator <- rep(0, m + 1)
    indicator[index] <- 1
    theta <- theta + gamma[iter]*(indicator - PI)

    #Cool the temperature (square root cooling rate).
    t <- initial_t/sqrt(iter)
  }


  if(HISTORY) {
    tour_history = tour_history[1:(changes-1), ]
  } else {
    tour_history = NULL
  }

  return(list(shortest_tour = tour_length(tour, cities),
              tour_history = tour_history,
              theta = theta))
}



# ######################################################
#
# Simulations.
```

```
#
# ###################################################
#m simulations are performed; the shortest tour from SA and
# SAA will be saved.
m <- 100
shortest_tours <- matrix(0, nrow = m, ncol = 2)

########################################
# Perform one simulation and store graphs.
########################################
n <- 100
cities <- generate_cities(n)

#Run SA and create graphs.
SA_results <- SA(cities, tour = 1:n)

png("images/SA_TSP_history.png", 1800, 1800, res = 240)
par(mfrow = c(3, 3), mar = rep(2, 4))
for(index in ceiling(seq(1, nrow(SA_results$tour_history),
                         length.out = 9))) {
  tour_plot(SA_results$tour_history[index, ], cities, axes = FALSE)
  print(SA_results$shortest_tour)
}
dev.off()

#Run SAA and create graphs.
SAA_results <- SAA(cities, tour = 1:n)

png("images/SAA_TSP_history.png", 1800, 1800, res = 240)
par(mfrow = c(3, 3), mar = rep(2, 4))
for(index in ceiling(seq(1, nrow(SAA_results$tour_history),
                         length.out = 9))) {
  tour_plot(SAA_results$tour_history[index, ], cities, axes = FALSE)
  print(tour_length(SAA_results$tour_history[index, ], cities))
}
dev.off()

shortest_tours[1, ] <- c(SA_results$shortest_tour, SAA_results$shortest_tour)

###################################
# Perform m - 1 more simulations.
###################################
for(i in 2:m) {
  shortest_tours[i, 1] <- (SA(generate_cities(n), tour = 1:n,
                              HISTORY = FALSE))$shortest_tour
  shortest_tours[i, 2] <- (SAA(generate_cities(n), tour = 1:n,
                               HISTORY = FALSE))$shortest_tour
```

```
}

write.csv(shortest_tours, "shortest_tours_10^5_iterations")
library("ggplot2")
colnames(shortest_tours) <- c("SA", "SAA")

par(mfrow = c(1, 1), mar = c(5, 4, 4, 2) + 1)
png("images/SA_vs_SAA.png", 400, 400, res = 100)
plot(shortest_tours[, 1], shortest_tours[, 2], pch = 16,
     main = "Shortest paths from in 100 simulations",
     xlab = "SA", ylab = "SAA")
abline(v = mean(shortest_tours[, 1]), col = "red")
abline(h = mean(shortest_tours[, 2]), col = "red")
dev.off()
```

References

Gelfand, A. E., Hills, S. E., Racine-Poon, A., & Smith, A. F. M. (1990). Illustration of bayesian inference in normal data models using gibbs sampling. *Journal of the American Statistical Association*, *85*(412), 972–985.

Liang, F., Cheng, Y., & Lin, G. (2014). Simulated stochastic approximation annealing for global optimization with a square-root cooling schedule. *Journal of the American Statistical Association*, *109*(506), 847–863. https://doi.org/10.1080/01621459.2013.872993

Yu, K., Liang, F., Ciampa, J., & Chatterjee, N. (2011). Efficient p-value evaluation for resampling-based tests. *Biostatistics*, *12*(3), 582–593. https://doi.org/10.1093/biostatistics/kxq078