Computational Statistics - Homework 1

Leon Löppert, Jan Parlesak, Paul Jarschke

04.06.2024

Problem 1

Problem 1.1: Conjugate Gradient Algorithm

```r
cg <- function(A, b, x) {
  # Initializations:
  r <- b - A %*% x  # initial residual vector
  p <- r  # direction vector
  j <- 0  # iteration counter

  conv <- c()  # convergence criterion tracker
  err <- c()  # error tracker

  conv[1] <- norm(r, "2")

  if (all(b == 0)) {
    # if b is a zero vector, track error
    err[1] <- norm(x, "I")
  }

  # Iterate until residual norm is sufficiently small
  # or reached max number of iterations
  cat("Starting Conguate Gradient algorithm...\n")
  while ((norm(r, "2") / norm(b, "2") > 1e-14) && j < 500) {
    alpha <- (crossprod(r)) / (t(p) %*% A %*% p)  # step size
    x <- x + c(alpha) * p  # Update solution vector x
    rnew <- r - c(alpha) * (A %*% p)  # new residual vector
    beta <-
      crossprod(rnew) / crossprod(r)  # calculate beta coefficient
    p <- rnew + c(beta) * p  # update direction vector
    r <- rnew  # update residual vector
    j <- j + 1  # update iteration counter
    conv[j] <- norm(r, "2") / norm(b, "2")  # track convergence

    if (all(b == 0)) {
      # if b is a zero vector, track error
      err[j] <- norm(x, "I")
    }
    # Iteration counter:
    cat(sprintf('It. %3.0f: %20.16e\n', j, conv[j]), "\n")
```

```r
  }
  cat("...finished!\n")
  cat("The solution for x is:", x, "\n")

  # Convergence plots (Convergence criterion over iterations):
  par(mfrow = c(2, 1))

  plot(
    1:j,
    conv,
    type = "o",
    col = "blue",
    pch = 16,
    cex = 0.5,
    main = "Convergence Plot",
    xlab = "Iteration",
    ylab = "Convergence Criterion"
  )
  lines(1:j, conv, lty = 2, col = 'blue')

  plot(
    1:j,
    conv,
    log = "y",
    type = "o",
    col = "red",
    pch = 16,
    cex = 0.5,
    main = "Convergence Plot",
    xlab = "Iteration",
    ylab = "Log Convergence Criterion"
  )
  lines(1:j, conv, lty = 2, col = 'red')

  # Return the convergence history and the final solution vector
  return(list(conv = conv, x = c(x)))
}
```

Problem 1.2: Preconditioned Conjugate Gradient Algorithm

```r
pcg <- function(A, b, x, M, tol = 1e-14) {
  # Initializations: ----
  r <- b - A %*% x  # initial residual vector
  z <- solve(M, r)  # apply preconditioner M to the residual vector
  p <- z  # direction vector
  j <- 1  # iteration counter

  conv <- c()  # convergence criterion tracker
  err <- c()  # error tracker

  conv[1] <- norm(r, "2")
```

```r
if (all(b == 0)) {
  # if b is a zero vector, track error
  err[1] <- norm(x, "I")
}

# Iterate until residual norm is sufficiently small
# or reached max number of iterations
cat("Starting Preconditioned Conguate Gradient algorithm...\n")

while ((norm(r, "2") / norm(b, "2") > tol) && j < 500) {
  alpha <- crossprod(z, r) / (t(p) %*% A %*% p)  # step size
  x <- x + c(alpha) * p  # Update solution vector x
  rnew <- r - c(alpha) * (A %*% p)  # new residual vector
  znew <- solve(M, rnew)  # update z
  beta <-
    crossprod(znew, rnew) / crossprod(z, r)  # calculate beta coefficient
  p <- znew + c(beta) * p  # update direction vector
  r <- rnew  # update residual vector
  z <- znew  # update z
  j <- j + 1  # update iteration counter

  conv[j] <- norm(r, "2") / norm(b, "2")  # track convergence

  if (all(b == 0)) {
    # if b is a zero vector, track error
    err[j] <- norm(x, "I")
  }
  # Iteration counter:
  cat(sprintf('It. %3.0f: %20.16e\n', j, conv[j]), "\n")
}
cat("...finished!\n")
cat("The solution for x is:", x, "\n")

# Convergence plots (Convergence criterion over iterations):
plot(
  1:j,
  conv,
  type = "o",
  col = "blue",
  pch = 16,
  cex = 0.5,
  main = "Convergence Plot",
  xlab = "Iteration",
  ylab = "Convergence Criterion"
)
lines(1:j, conv, lty = 2, col = "blue")

plot(
  1:j,
  conv,
  type = "o",
  col = "red",
  pch = 16,
```

```
    cex = 0.5,
    main = "Convergence Plot",
    xlab = "Iteration",
    ylab = "Log Convergence Criterion"
  )
  lines(1:j, conv, lty = 2, col = "red")

  # Return the convergence history and the final solution vector
  return(list(conv = conv, x = c(x)))
}
```

Problem 2

Problem 2.1: Modified Lanczos that creates A-orthonormal bases

```
a_lanczos <-
  function(A,  # Input matrix
           v0,  # Initial vector
           max_it = NULL,  # Maximum number of iterations
           tol = 1e-10,  # Tolerance value
           reorth = TRUE) {  # Flag for reorthogonalization

    # Preliminary calculations and initializations: ----
    N <- length(v0)

    if(!is.null(max_it)) max_it <- N

    # Initialize the basis matrix V with zeros
    V <- matrix(0, nrow = N, ncol = max_it + 1)

    # Compute initial vector Av and its norm beta
    Av <- A(v0)
    beta <- sqrt(sum(v0 * Av))

    # Set the first basis vector
    V[, 2] <-
      v0 / beta  # This is index 2 intentionally, 1 not returned

    # Normalize the initial vector Av
    Av <- Av / beta

    i <- 2  # Iteration counter

    # Lanczos iteration loop
    while (i < (max_it + 1) && beta > tol) {
      # Lanczos recursions
      w <- Av - beta * V[, i - 1]
      alpha <- sum(w * Av)
      w <- w - alpha * V[, i]

      # Reorthogonalization step
```

```
    if (reorth) {
      # Subtract projections onto previous basis vectors (twice)
      w <- w - (V[, 2:i] %*% crossprod(V[, 2:i], A(w)))
      w <- w - (V[, 2:i] %*% crossprod(V[, 2:i], A(w)))
    }

    # Norm of New Vector Av
    Av <- A(w)
    beta <- sqrt(sum(w * Av))

    # Store new vector if its norm is above the tolerance
    if (beta > tol) {
      i <- i + 1
      Av <- Av / beta
      V[, i] <- w / beta
    }
  }

  # Return the matrix V containing the A-orthonormal basis vectors
  return(V[, 2:i])
}
```

Problem 2.2: Bayesian Conjugate Gradient Method

```
bayescg <- function(A, b, x, Sig, max_it = NULL, tol = 1e-6, delay = NULL,
                    reorth = TRUE, NormA = NULL, xTrue = NULL,
                    SqrtSigTranspose = NULL) {

  ## Variable definitions: ----

  # Size of the system
  N <- length(x)

  # Default Maximum Iterations
  if (is.null(max_it)) {
    max_it <- N
  }

  # Residual and first search direction
  r <- matrix(0, nrow = N, ncol = max_it + 1)
  r[, 1] <- b - A(x)
  S <- r

  # Inner products
  rIP <- numeric(max_it + 1)
  rIP[1] <- sum(r[, 1] * r[, 1])
  sIP <- numeric(max_it)

  # Array holding matrix-vector products
  SigAs_hist <- matrix(0, nrow = N, ncol = max_it)
```

```r
# Convergence information
# If xTrue is supplied, more information is computed
rNorm <- sqrt(rIP[1])
Res2 <- numeric(max_it + 1)
if (is.null(NormA) || is.null(xTrue)) {
  bNorm <- norm(b, type = "2")
  Res <- rNorm / bNorm
  Res2[1] <- Res
}

if (!is.null(xTrue)) {
  xNorm <- norm(xTrue, type = "2")
  err_hist <- numeric(max_it + 1)
  err_hist[1] <- sum((x - xTrue) * A(x - xTrue))
  if (!is.null(NormA)) {
    xNormANorm <- norm(xTrue, type = "2") * NormA
    Res <- rNorm / xNormANorm
    Res2[1] <- Res
  }
  Res3 <- Res2
  tr_hist <- numeric(max_it + 1)
  tr_hist[1] <- sum(diag(A(Sig(diag(N)))))
}

i <- 0


## Iterating Through Bayesian Conjugate Gradient: ----

while (i < max_it && (is.null(tol) || Res > tol)) {
  # print(paste("Iteration:", i + 1))
  # Compute Matrix Vector Products
  As <- A(S[, i + 1])
  if (!is.null(SqrtSigTranspose)) {
    SigAs_hist[, i + 1] <- SqrtSigTranspose(As)
    SigAs <- Sig(SigAs_hist[, i + 1])
  } else {
    SigAs_hist[, i + 1] <- Sig(As)
    SigAs <- SigAs_hist[, i + 1]
  }
  ASigAs <- A(SigAs)

  # Search Direction Inner Product
  sIP[i + 1] <- abs(sum(S[, i + 1] * ASigAs))

  # Calculate next x
  alpha <- rIP[i + 1] / sIP[i + 1]
  x <- x + alpha * SigAs

  # Calculate New Residual
  r[, i + 2] <- r[, i + 1] - alpha * ASigAs

  if (reorth) {
```

```r
    # Reorthogonalize Residual
    r_ip_inv <- 1 / rIP[1:(i + 1)]
    ortho_term <- r[, 1:(i + 1)] %*% (t(r[, 1:(i + 1)]) %*% r[, i + 2])
    diag_r_ip_inv <- diag(r_ip_inv, nrow = length(r_ip_inv), ncol = length(r_ip_inv))
    if (ncol(ortho_term) == nrow(diag_r_ip_inv)) {
      r[, i + 2] <- r[, i + 2] - ortho_term %*% diag_r_ip_inv
    } # else {
      # if dimension checking is not successful, give warning
      # warning("Dimensions of ortho_term and diag(r_ip_inv) do not match")
    # }
  }

  # Compute Residual Norms
  rIP[i + 2] <- sum(r[, i + 2] * r[, i + 2])
  rNorm <- sqrt(rIP[i + 2])
  if (!is.null(xTrue)) {
    err_hist[i + 2] <- sum((x - xTrue) * A(x - xTrue))
    tr_hist[i + 2] <- tr_hist[i + 1] - sum(diag(A(outer(SigAs, SigAs)))) / sIP[i + 1]

    rTrueNorm <- norm(b - A(x), type = "2")
    if (!is.null(NormA)) {
      Res <- rNorm / xNormANorm
      Res3[i + 2] <- rTrueNorm / xNormANorm
    } else {
      Res3[i + 2] <- rTrueNorm / bNorm
    }
  }
  if (is.null(NormA)) {
    Res <- rNorm / bNorm
  } else if (is.null(xTrue)) {
    Res <- rNorm / NormA / norm(x, type = "2")
  }
  Res2[i + 2] <- Res

  # Calculate next search direction
  beta <- rIP[i + 2] / rIP[i + 1]
  S[, i + 2] <- r[, i + 2] + beta * S[, i + 1]

  i <- i + 1
}

## Return results: ----

info <- list(res = Res2[1:(i + 1)], search_dir = (sIP[1:i] ^ (-1/2)) * S[, 1:i])

if (!is.null(xTrue)) {
  info$actual_res <- Res3[1:(i + 1)]
  info$err <- err_hist[1:(i + 1)]
  info$trace <- tr_hist[1:(i + 1)]
}

if (!is.null(delay)) {
  delay <- min(delay, i)
```

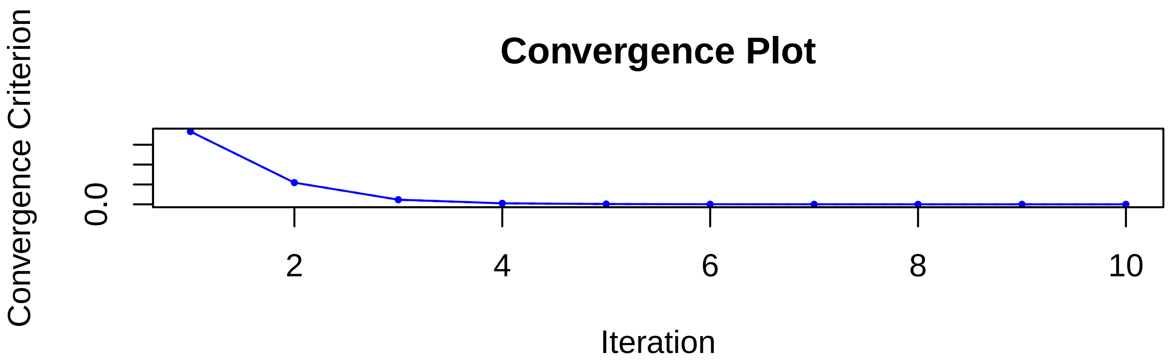
```r
    post_scale <- sum(rIP[(i - delay + 1):i]^2 / sIP[(i - delay + 1):i])
    info$scale <- post_scale
  }

  return(list(x = x, SigAs_hist = (sIP[1:i] ^ (-1/2)) * SigAs_hist[, 1:i], info = info))
}
```

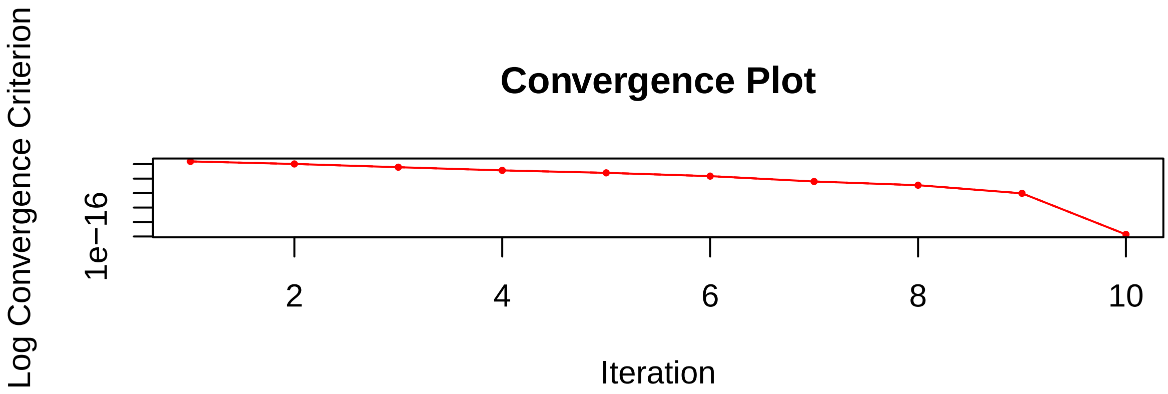
Examples

```r
# Set seed for reproducability
set.seed(123)

# Number of rows in linear system of equations
n <- 10

# Create a random symmetric positive-definite coefficient matrix A
createPSDmatrixA <-
  function(n) {
    A <- matrix(rnorm(n * n), n, n)
    A <- crossprod(A) + n * diag(n)
  }
A <- createPSDmatrixA(n)

# Generate a random vector b
b <- rnorm(n)

# Initial guess for x
x0 <- rep(0, n)

# Find solution using Conjugate Gradient Algorithm (Problem 1.1) ----
cg_results <- cg(A, b, x0)
```

```
## Starting Conguate Gradient algorithm...
## It.   1: 3.6631829446286168e-01
##
## It.   2: 1.0921528200647702e-01
##
## It.   3: 2.3227855416896950e-02
##
## It.   4: 5.1040155887402379e-03
##
## It.   5: 1.5745767128965691e-03
##
## It.   6: 3.2836615942801904e-04
##
## It.   7: 2.5295829214168488e-05
##
## It.   8: 4.2926878954089621e-06
##
## It.   9: 8.9040133058140012e-08
##
```

```
## It.  10: 2.6955219863623281e-16
##
## ...finished!
## The solution for x is: -0.03626387 0.0320711 -0.008359597 -0.02779354 -0.06587811 0.006584234 -0.046
```

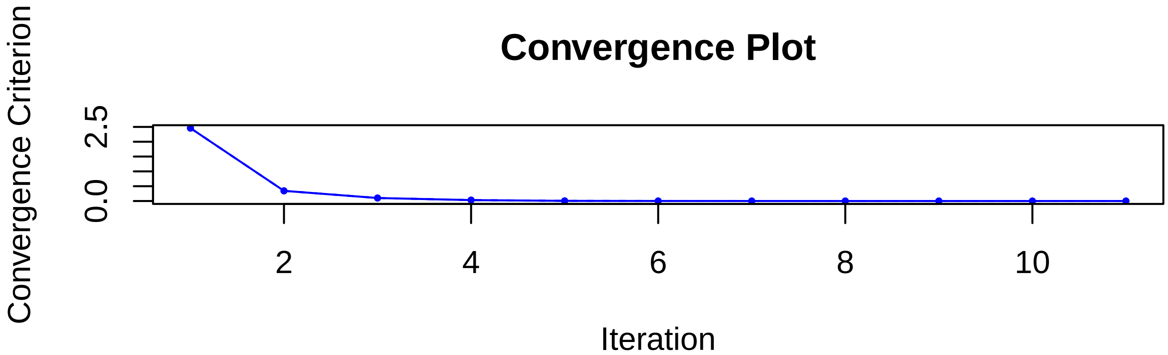


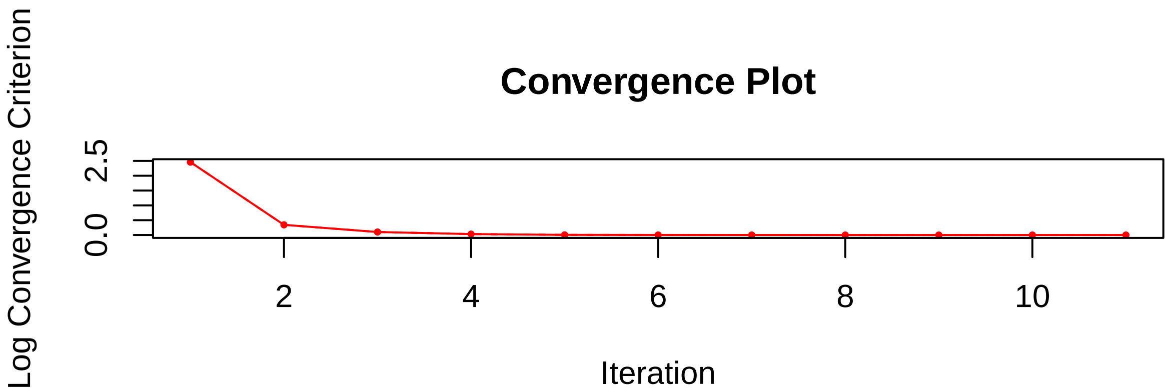



```r
# Find solution using Preconditioned Conjugate Gradient Algorithm (Problem 1.2) ----
M1 <- diag(diag(A))  # Jacobi preconditioner
M2 <- diag(n)  # Identity matrix

pcg_results1 <- pcg(A, b, x0, M1)
```

```
## Starting Preconditioned Conguate Gradient algorithm...
## It.   2: 3.4342927025576309e-01
##
## It.   3: 1.0103660141345808e-01
##
## It.   4: 3.1265737940853695e-02
##
## It.   5: 5.0008667290225781e-03
##
## It.   6: 1.4413296562075196e-03
##
## It.   7: 1.7965147376705902e-04
##
## It.   8: 5.2519637474155090e-05
```

```
##
## It.   9: 1.8864624590058102e-06
##
## It.  10: 3.0535807772870021e-07
##
## It.  11: 3.2476221894550656e-17
##
## ...finished!
## The solution for x is: -0.03626387 0.0320711 -0.008359597 -0.02779354 -0.06587811 0.006584234 -0.046
```

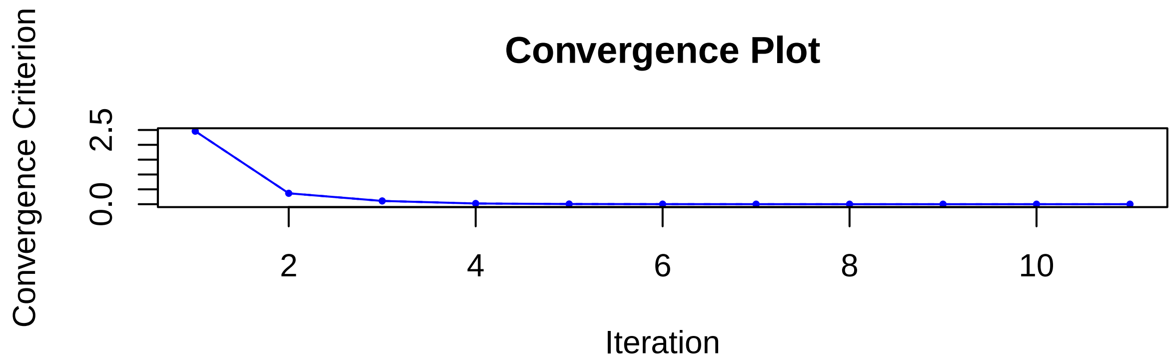


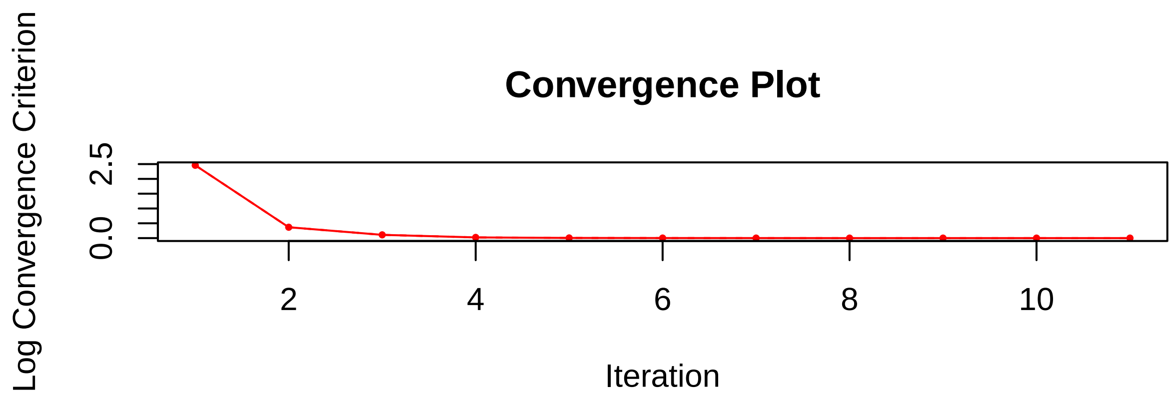



```r
pcg_results2 <- pcg(A, b, x0, M2)
```

```
## Starting Preconditioned Conguate Gradient algorithm...
## It.   2: 3.6631829446286168e-01
##
## It.   3: 1.0921528200647702e-01
##
## It.   4: 2.3227855416896950e-02
##
## It.   5: 5.1040155887402379e-03
##
## It.   6: 1.5745767128965691e-03
##
## It.   7: 3.2836615942801904e-04
##
```

```
## It.   8: 2.5295829214168488e-05
##
## It.   9: 4.2926878954089621e-06
##
## It.  10: 8.9040133058140012e-08
##
## It.  11: 2.6955219863623281e-16
##
## ...finished!
## The solution for x is: -0.03626387 0.0320711 -0.008359597 -0.02779354 -0.06587811 0.006584234 -0.0465
```






```r
# Testing a_lanczos function (Problem 2.1) ----
Sig <- function(v) v  # Define the preconditioner function Sig as the identity function
A_func <- function(v) A %*% v  # Define the function A(v) to apply the matrix A to a vector v
v0 <- rnorm(n)
lanczos_vectors <- a_lanczos(A_func, v0, max_it = 10, tol = 1e-10, reorth = TRUE)
print("Lanczos vectors:")
```

```
## [1] "Lanczos vectors:"
```

```r
lanczos_vectors
```

```
##                [,1]         [,2]        [,3]         [,4]          [,5]
##  [1,] -0.051841140  0.101155612  0.06783553 -0.085010108 -0.0070868978
```

```
##  [2,]  0.054780099  0.036835833  0.01801885  0.135973765 -0.0299798141
##  [3,] -0.145777922  0.038424041 -0.06509586  0.024184744 -0.0009181462
##  [4,] -0.005006363  0.027183998  0.05373108 -0.142822157  0.0696802424
##  [5,]  0.046800737  0.023321466 -0.10005378 -0.067802672 -0.0511211261
##  [6,]  0.027135163  0.123270156 -0.13412354  0.091139778  0.0648932751
##  [7,]  0.009521862 -0.071981089  0.06618394  0.027178599  0.1347912637
##  [8,] -0.057730261  0.110521848  0.02309881  0.052377238 -0.0571474350
##  [9,] -0.076561875  0.064939400 -0.05725563 -0.007506557  0.1291959813
## [10,] -0.092278239  0.004469559 -0.04433263  0.091370324 -0.0846999523
##              [,6]         [,7]        [,8]         [,9]        [,10]
##  [1,]  0.02322114 -0.201846517  0.03549281  0.025781726  0.009292058
##  [2,] -0.01916601 -0.021334494 -0.17278404 -0.036957540  0.058224245
##  [3,]  0.08010360  0.005285938 -0.07371684  0.002812149  0.157201242
##  [4,] -0.11216775  0.125953413 -0.03405200 -0.088633611  0.130300575
##  [5,]  0.05867997 -0.068890568 -0.03696493 -0.152851145 -0.034916536
##  [6,] -0.02468506 -0.011782276  0.10677914  0.015989472  0.092718176
##  [7,]  0.03452520 -0.077879904  0.06930027 -0.148910802  0.024526129
##  [8,]  0.06436904  0.128326535  0.04796898 -0.085780914 -0.086232906
##  [9,] -0.06077777  0.016042388 -0.09816357  0.028176329 -0.184069967
## [10,] -0.10795388 -0.054664030  0.06705214 -0.093956532 -0.007310916
```

```r
# Find solution using the Bayesian Conjugate Gradient Algorithm (Problem 2.2) ----
bayescg_results <- bayescg(A_func, b, x0, Sig, max_it = 10, tol = 1e-6, reorth = TRUE)

# Find solution using implemented R solver ----
R_solver_results <- solve(A, b)

# Compare results ----
cat('Final solutions for x:\n')
```

```
## Final solutions for x:
```

```r
cat('... using Conjugate Gradient Algorithm:\n')
```

```
## ... using Conjugate Gradient Algorithm:
```

```r
cg_results$x
```

```
##  [1] -0.036263867  0.032071103 -0.008359597 -0.027793538 -0.065878106
##  [6]  0.006584234 -0.046571887 -0.082618303 -0.033472677  0.018976913
```

```r
cat('... using Preconditioned Conjugate Gradient Algorithm (Jacobi):\n')
```

```
## ... using Preconditioned Conjugate Gradient Algorithm (Jacobi):
```

```r
pcg_results1$x
```

```
##  [1] -0.036263867  0.032071103 -0.008359597 -0.027793538 -0.065878106
##  [6]  0.006584234 -0.046571887 -0.082618303 -0.033472677  0.018976913
```

```r
cat('... using Preconditioned Conjugate Gradient Algorithm (Identity):\n')
```

```
## ... using Preconditioned Conjugate Gradient Algorithm (Identity):
```

```r
pcg_results2$x
```

```
##  [1] -0.036263867  0.032071103 -0.008359597 -0.027793538 -0.065878106
##  [6]  0.006584234 -0.046571887 -0.082618303 -0.033472677  0.018976913
```

```r
cat('... using Bayesian Conjugate Gradient Algorithm:\n')
```

```
## ... using Bayesian Conjugate Gradient Algorithm:
```

```r
bayescg_results$x
```

```
##  [1] -0.036263867  0.032071103 -0.008359597 -0.027793538 -0.065878106
##  [6]  0.006584234 -0.046571887 -0.082618303 -0.033472677  0.018976913
```

```r
cat('... using R Solver:\n')
```

```
## ... using R Solver:
```

```r
R_solver_results
```

```
##  [1] -0.036263867  0.032071103 -0.008359597 -0.027793538 -0.065878106
##  [6]  0.006584234 -0.046571887 -0.082618303 -0.033472677  0.018976913
```

```r
all.equal(cg_results$x, R_solver_results, tolerance = 1e-5)
```

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

```r
all.equal(pcg_results1$x, R_solver_results, tolerance = 1e-5)
```

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

```r
all.equal(pcg_results2$x, R_solver_results, tolerance = 1e-5)
```

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

```r
all.equal(bayescg_results$x, R_solver_results, tolerance = 1e-5)
```

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