Solutions to Exercises

Portfolio Optimization: Theory and Application
Appendix B – Optimization Algorithms

Daniel P. Palomar (2025). *Portfolio Optimization: Theory and Application.* Cambridge University Press.

portfoliooptimizationbook.com

Contributors:

- Runhao SHI
- Daniel Palomar

Exercise B.1: Euclidean norm approximation

a. Randomly generate the parameters $\boldsymbol{A} \in \mathbb{R}^{10\times 5}$ and $\boldsymbol{b} \in \mathbb{R}^{10}$.
b. Formulate a regression problem to approximate $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$ based on the ℓ_2-norm.
c. Solve it directly with the least squares closed-form solution.
d. Solve it using a modeling framework (e.g., CVX).
e. Solve it invoking a QP solver.

Solution

TBD

Exercise B.2: Manhattan norm approximation

a. Randomly generate the parameters $\boldsymbol{A} \in \mathbb{R}^{10\times 5}$ and $\boldsymbol{b} \in \mathbb{R}^{10}$.
b. Formulate a regression problem to approximate $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$ based on the ℓ_1-norm.
c. Solve it using a modeling framework (e.g., CVX).

d. Rewrite it as an LP and solve it invoking an LP solver.

Solution

TBD

Exercise B.3: Chebyshev norm approximation

a. Randomly generate the parameters $\boldsymbol{A} \in \mathbb{R}^{10\times 5}$ and $\boldsymbol{b} \in \mathbb{R}^{10}$.
b. Formulate a regression problem to approximate $\boldsymbol{Ax} \approx \boldsymbol{b}$ based on the ℓ_∞-norm.
c. Solve it using a modeling framework (e.g., CVX).
d. Rewrite it as an LP and solve it invoking an LP solver.

Solution

TBD

Exercise B.4: Solving an LP

Consider the following LP:

$$\begin{array}{ll} \underset{x_1,x_2}{\text{maximize}} & 3x_1 + x_2 \\ \text{subject to} & x_1 + 2x_2 \leq 4, \\ & 4x_1 + 2x_2 \leq 12, \\ & x_1, x_2 \geq 0. \end{array}$$

a. Solve it using a modeling framework (e.g., CVX).
b. Solve it by directly invoking an LP solver.
c. Solve it by invoking a general-purpose nonlinear solver.
d. Implement the projected gradient method to solve the problem.
e. Implement the constrained Newton's method to solve the problem.
f. Implement the log-barrier interior-point method to solve the problem (use (1,1) as the initial point).
g. Compare all the solutions and the computation time.

Solution

TBD

Exercise B.5: Central path

Formulate the log-barrier problem corresponding to the LP in Exercise B.4 and plot the central path as the parameter t varies.

Solution

TBD

Exercise B.6: Phase I method

Design a phase I method to find a feasible point for the LP in Exercise B.4, which can then be used as the starting point for the barrier method.

Solution

TBD

Exercise B.7: Dual problem

Formulate the dual problem corresponding to the LP in Exercise B.4 and solve it using a solver of your choice.

Solution

TBD

Exercise B.8: KKT conditions

Write down the Karush–Kuhn–Tucker (KKT) conditions for the LP in Exercise B.4 and discuss their role in determining the optimality of a solution.

Solution

TBD

Exercise B.9: Solving a QP

Consider the following QP:

$$\begin{array}{ll} \underset{x_1, x_2}{\text{maximize}} & x_1^2 + x_2^2 \\ \text{subject to} & x_1 + x_2 = 1, \\ & x_1 \geq 0, x_2 \geq 0. \end{array}$$

a. Solve it using a modeling framework (e.g., CVX).
b. Solve it by directly invoking a QP solver.
c. Solve it by invoking a general-purpose nonlinear solver.
d. Implement the projected gradient method to solve the problem.
e. Implement the constrained Newton's method to solve the problem.
f. Implement the log-barrier interior-point method to solve the problem (use (0.5,0.5) as the initial point).
g. Compare all the solutions and the computation time.

Solution

TBD

Exercise B.10: Fractional programming

Consider the following fractional program:

$$\begin{array}{ll} \underset{\boldsymbol{w}}{\text{maximize}} & \dfrac{\boldsymbol{w}^\mathsf{T}\mathbf{1}}{\sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{w}}} \\ \text{subject to} & \mathbf{1}^\mathsf{T}\boldsymbol{w} = 1, \quad \boldsymbol{w} \geq \mathbf{0}, \end{array}$$

where $\boldsymbol{\Sigma} \succ \mathbf{0}$.

a. Solve it with a general-purpose nonlinear solver.
b. Solve it via bisection.
c. Solve it via the Dinkelbach method as a sequence of SOCPs.
d. Develop a modified algorithm that solves the problem as a sequence of QPs instead.
e. Solve it via the Schaible transform method.
f. Reformulate the problem as a minimization and then solve it via the Schaible transform method.
g. Compare all the previous approaches in terms of the accuracy of the solution and the computation time.

Solution

TBD

Exercise B.11: Soft-thresholding operator

Consider the following convex optimization problem:

$$\underset{x}{\text{minimize}} \quad \tfrac{1}{2}\|\boldsymbol{a}x - \boldsymbol{b}\|_2^2 + \lambda|x|,$$

with $\lambda \geq 0$. Derive the solution and show that it can be written as

$$x = \frac{1}{\|\boldsymbol{a}\|_2^2}\mathcal{S}_\lambda\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{b}\right),$$

where $\mathcal{S}_\lambda(\cdot)$ is the so-called soft-thresholding operator defined as

$$\mathcal{S}_\lambda(u) = \text{sign}(u)(|u| - \lambda)^+,$$

with $\text{sign}(\cdot)$ denoting the sign function and $(\cdot)^+ = \max(0, \cdot)$.

Solution

There are three possible options for the optimal solution:

- $x > 0$: the objective (ignoring a constant term) becomes $\frac{1}{2}\|\boldsymbol{a}\|_2^2x^2 - \boldsymbol{a}^\mathsf{T}\boldsymbol{b}x + \lambda x$ and setting the derivative to zero leads to
$$x = \frac{1}{\|\boldsymbol{a}\|_2^2}\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{b} - \lambda\right)$$
which implies $\boldsymbol{a}^\mathsf{T}\boldsymbol{b} > \lambda \geq 0$;
- $x < 0$: the objective (ignoring a constant term) becomes $\frac{1}{2}\|\boldsymbol{a}\|_2^2x^2 - \boldsymbol{a}^\mathsf{T}\boldsymbol{b}x - \lambda x$ and setting

the derivative to zero leads to

$$x = \frac{1}{\|\boldsymbol{a}\|_2^2}\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{b} + \lambda\right)$$

which implies $\boldsymbol{a}^\mathsf{T}\boldsymbol{b} < -\lambda \leq 0$;

- $x = 0$: this is the last possible case, which can only happen when $\boldsymbol{a}^\mathsf{T}\boldsymbol{b} \in [-\lambda, \lambda]$ or, equivalently, $|\boldsymbol{a}^\mathsf{T}\boldsymbol{b}| \leq \lambda$.

The solution can be written as

$$x = \frac{1}{\|\boldsymbol{a}\|_2^2}\mathrm{sign}\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{b}\right)\left(|\boldsymbol{a}^\mathsf{T}\boldsymbol{b}| - \lambda\right)^+,$$

where

$$\mathrm{sign}(u) = \begin{cases} +1 & u > 0, \\ 0 & u = 0, \\ -1 & u < 0 \end{cases}$$

is the sign function and $(\cdot)^+ = \max(0, \cdot)$. This can be written more compactly as

$$x = \frac{1}{\|\boldsymbol{a}\|_2^2}\mathcal{S}_\lambda\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{b}\right),$$

where $\mathcal{S}_\lambda(\cdot)$ is the soft-thresholding operator.

Exercise B.12: ℓ_2–ℓ_1-norm minimization

Consider the following ℓ_2–ℓ_1-norm minimization problem (with $\boldsymbol{A} \in \mathbb{R}^{10\times 5}$ and $\boldsymbol{b} \in \mathbb{R}^{10}$ randomly generated):

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

a. Solve it using a modeling framework (e.g., CVX).
b. Rewrite the problem as a QP and solve it by invoking a QP solver.
c. Solve it with an ad hoc LASSO solver.

Solution

First, let's generate the data:

```r
# Generate data
set.seed(42)
lmd <- 2
m <- 500
n <- 100
A <- matrix(rnorm(m*n), m, n)
x_true <- rnorm(n)
b <- A %*% x_true + 0.1*rnorm(m)
```

a. Solution via CVX:

```r
library(CVXR)

# Get optimal value via CVX
x <- Variable(n)
prob <- Problem(Minimize(0.5*cvxr_norm(A %*% x - b, 2)^2 + lmd*cvxr_norm(x, 1)))
res <- solve(prob)
x_cvx <- res$getValue(x)
opt_value <- res$value
print(opt_value)
```

```
[1] 156.1089
```

```r
# Alternatively, the optimal value is:
print(0.5*sum((A %*% x_cvx - b)^2) + lmd*sum(abs(x_cvx)))
```

```
[1] 156.1089
```

b. Solution via QP solver. First, we rewrite the problem as

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} + \lambda\|\boldsymbol{x}\|_1$$

and then

$$\begin{array}{ll}\underset{\boldsymbol{x},\boldsymbol{t}}{\text{minimize}} & \tfrac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\mathsf{T}\boldsymbol{A}\boldsymbol{x} + \lambda\boldsymbol{1}^\mathsf{T}\boldsymbol{t}\\ \text{subject to} & -\boldsymbol{t} \leq \boldsymbol{x} \leq \boldsymbol{t}\end{array}$$

```r
library(CVXR)

# Get optimal value as a QP via CVX
x <- Variable(n)
t <- Variable(n)
prob <- Problem(Minimize(0.5*cvxr_norm(A %*% x, 2)^2 - t(b) %*% A %*% x + lmd*sum(t)),
                constraints = list(-t <= x, x <= t))
res <- solve(prob)
x_QP_cvx <- res$getValue(x)

# Sanity check:
print(0.5*sum((A %*% x_QP_cvx - b)^2) + lmd*sum(abs(x_QP_cvx)))
```

```
[1] 156.1108
```

```r
library(quadprog)

# Get optimal value as a QP via quadprog (z = [x; t])
P <- matrix(0, 2*n, 2*n)
P[1:n, 1:n] <- t(A) %*% A  # The quadratic term coefficient
q <- c(t(A) %*% b, rep(lmd, n))  # The linear term coefficients

# Constraint matrices for -t <= x <= t
A_constraint <- rbind(
  cbind(diag(n), -diag(n)),     # x - t <= 0
  cbind(-diag(n), -diag(n))     # -x - t <= 0
)
b_constraint <- rep(0, 2*n)

# Solve problem
result <- solve.QP(
  Dmat = P + diag(1e-8, nrow(P)),
  dvec = q,
  Amat = t(-A_constraint),  # Transpose and negate due to quadprog convention
  bvec = b_constraint,
  meq = 0  # No equality constraints
)

# Extract the solution
z_QP <- result$solution
x_QP <- z_QP[1:n]
t_QP <- z_QP[(n+1):(2*n)]

# Sanity check:
print(0.5*sum((A %*% x_QP - b)^2) + lmd*sum(abs(x_QP)))
```

```
[1] 156.603
```

c. Solution via ad hoc LASSO solver:

```r
library(glmnet)

# Perform LASSO regression (alpha=1 specifies LASSO, i.e., L1 penalty)
lasso_model <- glmnet(A, b, alpha = 1, lambda = lmd / m,
                      standardize = FALSE, intercept = FALSE)

# Extract the solution
x_lasso <- as.vector(coef(lasso_model))[-1]  # remove intercept

# Sanity check:
print(0.5*sum((A %*% x_lasso - b)^2) + lmd*sum(abs(x_lasso)))
```

```
[1] 156.1091
```

Exercise B.13: BCD for ℓ_2–ℓ_1-norm minimization

Solve the ℓ_2–ℓ_1-norm minimization problem in Exercise B.12 via BCD. Plot the convergence vs. iterations and CPU time.

Solution

We will use BCD by dividing the variable into each constituent element $\boldsymbol{x} = (x_1, \dots, x_n)$. Therefore, the sequence of problems at each iteration $k = 0, 1, 2, \dots$ for each element $i = 1, \dots, n$ is

$$\underset{x_i}{\text{minimize}} \quad \tfrac{1}{2}\left\|\boldsymbol{a}_i x_i - \tilde{\boldsymbol{b}}_i^k\right\|_2^2 + \lambda |x_i|,$$

where $\tilde{\boldsymbol{b}}_i^k \triangleq \boldsymbol{b} - \sum_{j<i} \boldsymbol{a}_j x_j^{k+1} - \sum_{j>i} \boldsymbol{a}_j x_j^k$.
This leads to the following iterative algorithm for $k = 0, 1, 2, \dots$

$$x_i^{k+1} = \frac{1}{\|\boldsymbol{a}_i\|_2^2} \mathcal{S}_\lambda\left(\boldsymbol{a}_i^\mathsf{T} \tilde{\boldsymbol{b}}_i^k\right), \qquad i = 1, \dots, n,$$

where $\mathcal{S}_\lambda(\cdot)$ is the soft-thresholding operator defined as

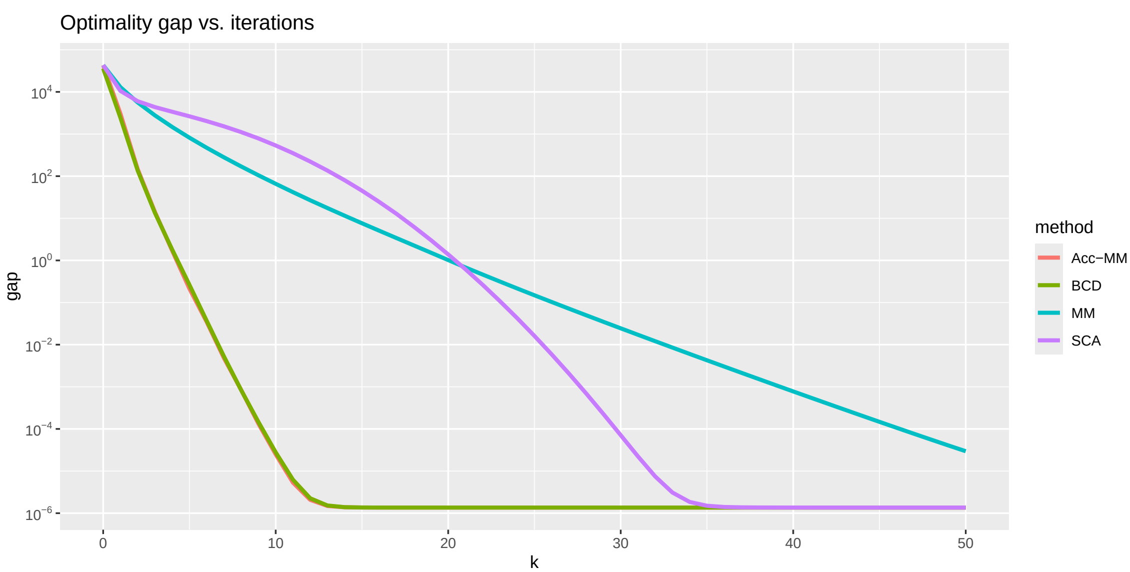
$$\mathcal{S}_\lambda(u) = \text{sign}(u)(|u| - \lambda)^+. \tag{1}$$

Convergence of BCD for the ℓ_2–ℓ_1-norm minimization:

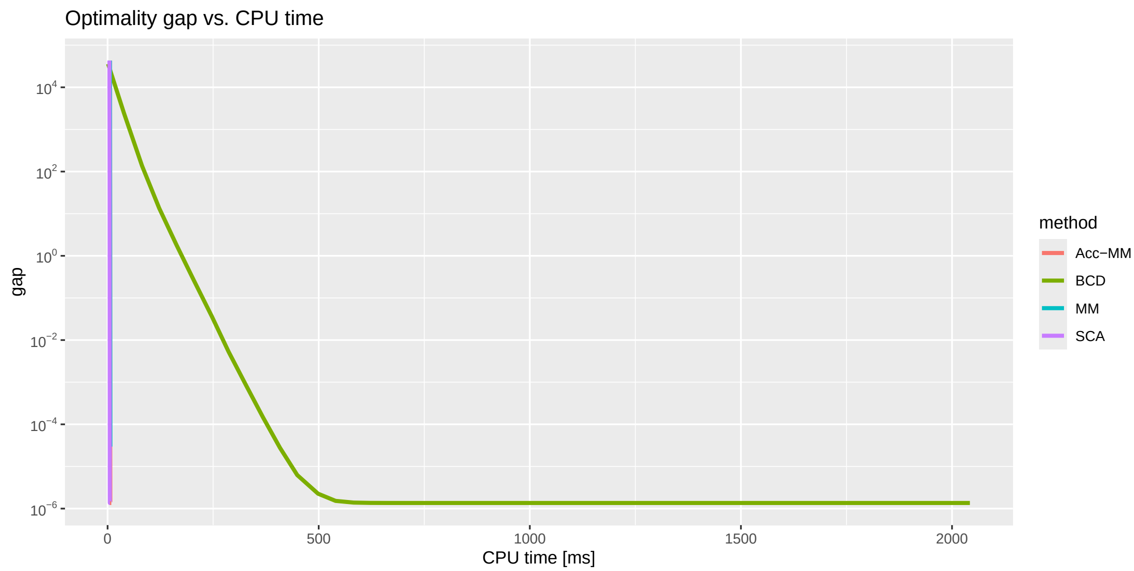
```r
library(microbenchmark)

# Set up
set.seed(42)
x0 <- rnorm(n)
soft_thresholding <- function(u, lmd) sign(u)*pmax(0, abs(u) - lmd)
num_times <- 10L  # to compute the cpu time


# Solve problem via BCD
x <- x0
cpu_time <- microbenchmark({
  a2 <- apply(A, 2, function(x) sum(x^2))  # initial overhead
  }, unit = "microseconds", times = num_times)$time |> median()
df <- data.frame(
  "k"          = 0,
  "cpu time k" = cpu_time,
  "gap"        = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
  "method"     = "BCD",
  check.names = FALSE
  )
for (k in 1:50) {
  cpu_time <- microbenchmark({
    x_new <- x
    for (i in 1:n) {
      b_ <- b - A[, -i] %*% x_new[-i]
      x_new[i] <- soft_thresholding(t(A[, i]) %*% b_, lmd)/a2[i]
      }
    }, unit = "microseconds", times = num_times)$time |> median()
  x <- x_new

  df <- rbind(df, list(
    "k"          = k,
    "cpu time k" = cpu_time,
    "gap"        = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
    "method"     = "BCD")
    )
}
```

Plot convergence:

```r
library(ggplot2)
library(dplyr)
library(scales)

# Compute cumulative CPU time over iterations
df <- df |>
  group_by(method) |>
  mutate("CPU time [ms]" = cumsum(`cpu time k`)/1e6) |>
  ungroup()

# Plots
df |>
  ggplot(aes(x = k, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  ggtitle("Optimality gap vs. iterations")
```




```r
df |>
  ggplot(aes(x = `CPU time [ms]`, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  ggtitle("Optimality gap vs. CPU time")
```



Exercise B.14: MM for ℓ_2–ℓ_1-norm minimization

Solve the ℓ_2–ℓ_1-norm minimization problem in Exercise B.12 via MM and its accelerated version. Plot the convergence vs. iterations and CPU time.

Solution

We can develop a simple iterative algorithm based on MM that leverages the element-by-element closed-form solution. One possible majorizer of the objective function $f(\boldsymbol{x})$ is

$$u\left(\boldsymbol{x};\boldsymbol{x}^k\right)=\frac{\kappa}{2}\|\boldsymbol{x}-\bar{\boldsymbol{x}}^k\|_2^2+\lambda\|\boldsymbol{x}\|_1+\text{constant},$$

where $\bar{\boldsymbol{x}}^k=\boldsymbol{x}^k-\frac{1}{\kappa}\boldsymbol{A}^\mathsf{T}\left(\boldsymbol{A}\boldsymbol{x}^k-\boldsymbol{b}\right)$.
Therefore, the sequence of majorized problems to be solved for $k=0,1,2,\dots$ is

$$\underset{\boldsymbol{x}}{\text{minimize}}\quad \tfrac{\kappa}{2}\left\|\boldsymbol{x}-\bar{\boldsymbol{x}}^k\right\|_2^2+\lambda\|\boldsymbol{x}\|_1,$$

which decouples into each component of $\boldsymbol{x}$ with closed-form solution given by the soft-thresholding operator.
This finally leads to the following MM iterative algorithm:

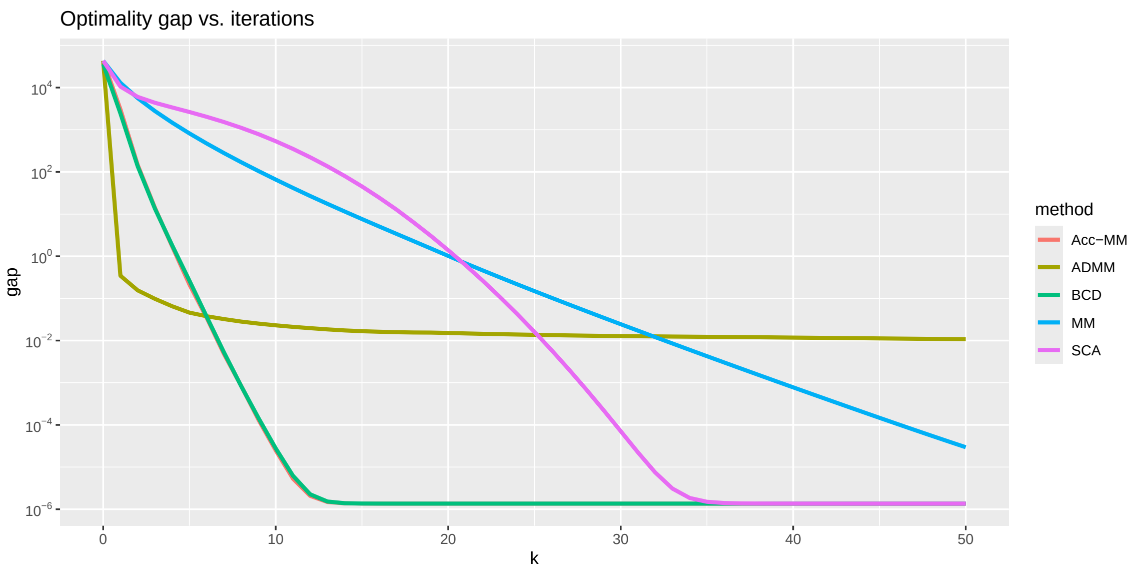
$$\boldsymbol{x}^{k+1}=\mathcal{S}_{\lambda/\kappa}\left(\bar{\boldsymbol{x}}^k\right),\qquad k=0,1,2,\dots,$$

where $\mathcal{S}_{\lambda/\kappa}(\cdot)$ is the soft-thresholding operator.
Convergence of MM for the ℓ_2–ℓ_1-norm minimization:

```r
# Set up
set.seed(42)
x0 <- rnorm(n)
soft_thresholding <- function(u, lmd) sign(u)*pmax(0, abs(u) - lmd)
num_times <- 10L  # to compute the cpu time


# Solve problem via MM
x <- x0
cpu_time <- microbenchmark({
  Atb <- t(A) %*% b
  AtA <- t(A) %*% A
  #kappa <- 1.1 * max(eigen(AtA)$values)
  u <- x0; for (i in 1:20) u <- AtA %*% u  # power iteration method
  kappa <- 1.1 * as.numeric(t(u) %*% AtA %*% u / sum(u^2))
  }, unit = "microseconds", times = num_times)$time |> median()
df <- rbind(df, list(
  "k"             = 0,
  "cpu time k"    = cpu_time,
  "CPU time [ms]" = NA,
  "gap"           = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
  "method"        = "MM")
  )
for (k in 1:50) {
  cpu_time <- microbenchmark({
    x_new <- soft_thresholding(x - (AtA %*% x - Atb)/kappa, lmd/kappa)
    }, unit = "microseconds", times = num_times)$time |> median()
  x <- x_new

  df <- rbind(df, list(
    "k"             = k,
    "cpu time k"    = cpu_time,
    "CPU time [ms]" = NA,
    "gap"           = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
    "method"        = "MM")
    )
}


# Accelerated MM
x <- x0
cpu_time <- microbenchmark({
  Atb <- t(A) %*% b
  AtA <- t(A) %*% A
  #kappa <- 1.1 * max(eigen(AtA)$values)
  u <- x0; for (i in 1:20) u <- AtA %*% u  # power iteration method
  kappa <- 1.1 * as.numeric(t(u) %*% AtA %*% u / sum(u^2))
  }, unit = "microseconds", times = num_times)$time |> median()
df <- rbind(df, list(
  "k"             = 0,
  "cpu time k"    = cpu_time,
  "CPU time [ms]" = NA,
  "gap"           = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
  "method"        = "Acc-MM")
  )
for (k in 1:50) {
  cpu_time <- microbenchmark({
    MM_x    <- soft_thresholding(x - (AtA %*% x - Atb)/kappa, lmd/kappa)
```

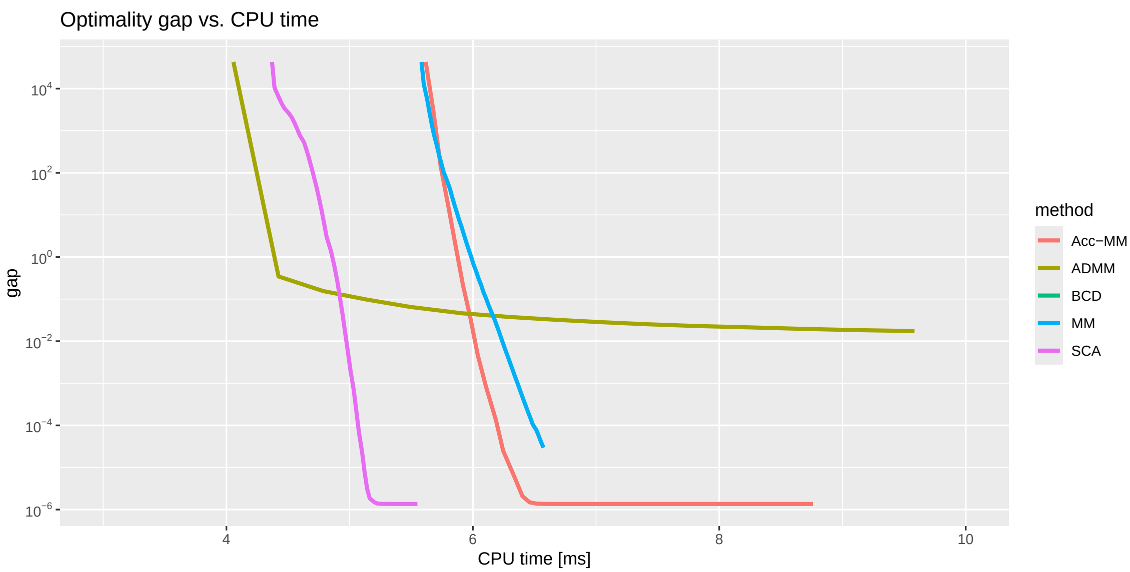
Plot convergence:

```
library(ggplot2)
library(dplyr)
library(scales)

# Compute cumulative CPU time over iterations
df <- df |>
  group_by(method) |>
  mutate("CPU time [ms]" = cumsum(`cpu time k`)/1e6) |>
  ungroup()

# Plots
df |>
  ggplot(aes(x = k, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  ggtitle("Optimality gap vs. iterations")
```




```
df |>
  ggplot(aes(x = `CPU time [ms]`, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  xlim(c(3, 10)) +
  ggtitle("Optimality gap vs. CPU time")
```

```
Warning: Removed 87 rows containing missing values or values outside the scale range
(`geom_line()`).
```



Exercise B.15: SCA for ℓ_2–ℓ_1-norm minimization

Solve the ℓ_2–ℓ_1-norm minimization problem in Exercise B.12 via SCA. Plot the convergence vs. iterations and CPU time.

Solution

We will now develop a simple iterative algorithm based on SCA that leverages the element-by-element closed-form solution. We can use parallel SCA by partitioning the variable $\boldsymbol{x}$ into each element $(x_1, \dots, x_n)$ and employing the surrogate functions

$$\tilde{f}\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right) = \frac{1}{2}\left\|\boldsymbol{a}_i x_i - \tilde{\boldsymbol{b}}_i^k\right\|_2^2 + \lambda |x_i| + \frac{\tau}{2}\left(x_i - x_i^k\right)^2,$$

where $\tilde{\boldsymbol{b}}_i^k = \boldsymbol{b} - \sum_{j\neq i} \boldsymbol{a}_j x_j^k$.
Therefore, the sequence of surrogate problems to be solved for $k = 0, 1, 2, \dots$ is

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{1}{2}\left\|\boldsymbol{a}_i x_i - \tilde{\boldsymbol{b}}_i^k\right\|_2^2 + \lambda |x_i| + \tau\left(x_i - x_i^k\right)^2, \qquad i = 1, \dots, n,$$

with the solution given by the soft-thresholding operator.

This finally leads to the following SCA iterative algorithm:

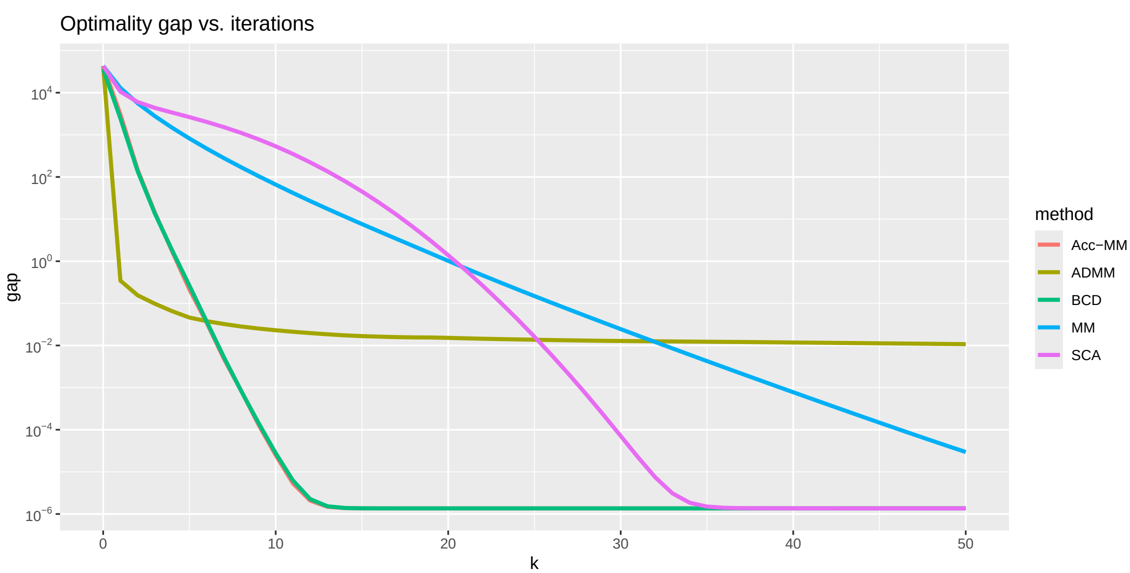
$$\left.\begin{aligned} \hat{x}_i^{k+1} &= \frac{1}{\tau + \|\boldsymbol{a}_i\|^2} \mathcal{S}_\lambda \left(\boldsymbol{a}_i^\mathsf{T} \tilde{\boldsymbol{b}}_i^k + \tau x_i^k \right) \\ x_i^{k+1} &= x_i^k + \gamma^k \left(\hat{x}_i^{k+1} - x_i^k \right) \end{aligned} \right\} \qquad i = 1, \ldots, n, \quad k = 0, 1, 2, \ldots,$$

where $\mathcal{S}_\lambda(\cdot)$ is the soft-thresholding operator.
Convergence of SCA for the ℓ_2–ℓ_1-norm minimization:

```r
# Set up
set.seed(42)
x0 <- rnorm(n)
soft_thresholding <- function(u, lmd) sign(u)*pmax(0, abs(u) - lmd)
num_times <- 10L  # to compute the cpu time


# Solve problem via SCA
tau <- 1e-6
eps <- 0.01
gamma <- 1
x <- x0
cpu_time <- microbenchmark({
  Atb <- t(A) %*% b
  AtA <- t(A) %*% A
  tau_plus_a2 <- tau + diag(AtA)
  }, unit = "microseconds", times = num_times)$time |> median()
df <- rbind(df, list(
  "k"             = 0,
  "cpu time k"    = cpu_time,
  "CPU time [ms]" = NA,
  "gap"           = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
  "method"        = "SCA")
  )
for (k in 1:50) {
  cpu_time <- microbenchmark({
    x_hat <- soft_thresholding(x - (AtA %*% x - Atb)/tau_plus_a2, lmd/tau_plus_a2)
    x_new <- gamma*x_hat + (1 - gamma)*x
    }, unit = "microseconds", times = num_times)$time |> median()
  x <- x_new
  gamma <- gamma * (1 - eps*gamma)

  df <- rbind(df, list(
    "k"          = k,
    "cpu time k" = cpu_time,
    "CPU time [ms]" = NA,
    "gap"        = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
    "method"     = "SCA")
    )
}
```

Plot convergence:

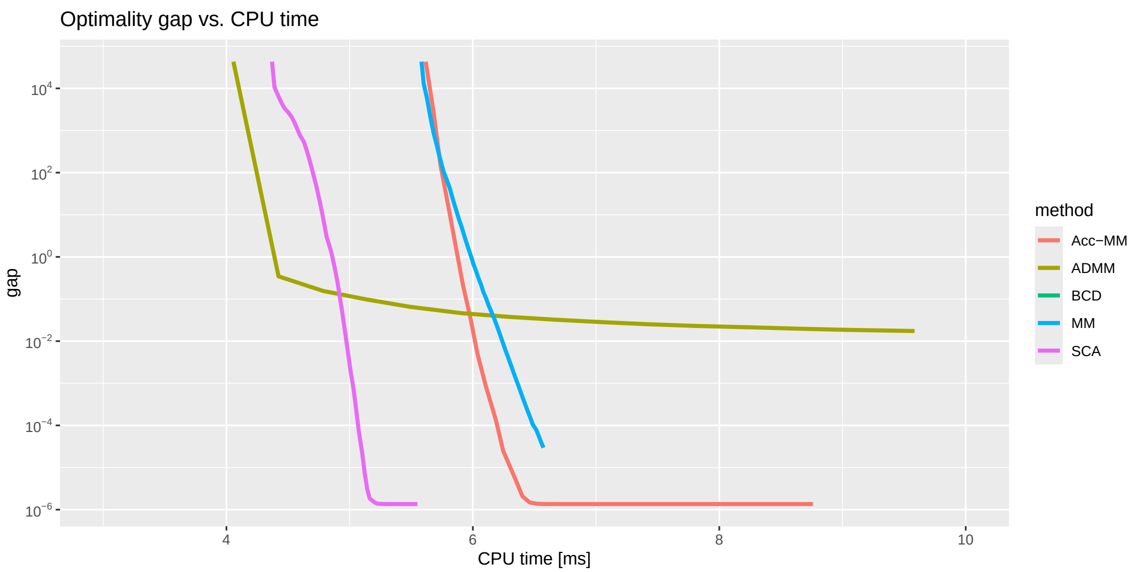
```r
library(ggplot2)
library(dplyr)
library(scales)

# Compute cumulative CPU time over iterations
df <- df |>
  group_by(method) |>
  mutate("CPU time [ms]" = cumsum(`cpu time k`)/1e6) |>
  ungroup()

# Plots
df |>
  ggplot(aes(x = k, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  ggtitle("Optimality gap vs. iterations")
```




```r
df |>
  ggplot(aes(x = `CPU time [ms]`, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  xlim(c(3, 10)) +
  ggtitle("Optimality gap vs. CPU time")
```

```
Warning: Removed 87 rows containing missing values or values outside the scale range
```

(`geom_line()`).



Exercise B.16: ADMM for ℓ_2–ℓ_1-norm minimization

Solve the ℓ_2–ℓ_1-norm minimization problem in Exercise B.12 via ADMM. Plot the convergence vs. iterations and CPU time.

Solution

We start by reformulating the problem as

$$\begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} & \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\lambda\|\boldsymbol{z}\|_1 \\ \text{subject to} & \boldsymbol{x}-\boldsymbol{z}=\boldsymbol{0}. \end{array}$$

The $\boldsymbol{x}$-update, given $\boldsymbol{z}$ and the scaled dual variable $\boldsymbol{u}$, is the solution to

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\frac{\rho}{2}\left\|\boldsymbol{x}-\boldsymbol{z}+\boldsymbol{u}\right\|_2^2,$$

given by $\boldsymbol{x}=\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A}+\rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b}+\rho(\boldsymbol{z}-\boldsymbol{u})\right)$, whereas the $\boldsymbol{z}$-update, given $\boldsymbol{x}$ and $\boldsymbol{u}$, is the solution to

$$\underset{\boldsymbol{z}}{\text{minimize}} \quad \frac{\rho}{2}\left\|\boldsymbol{x}-\boldsymbol{z}+\boldsymbol{u}\right\|_2^2+\lambda\|\boldsymbol{z}\|_1$$

given by $\boldsymbol{z}=\mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x}+\boldsymbol{u}\right)$, where $\mathcal{S}_{\lambda/\rho}(\cdot)$ is the soft-thresholding operator.

Thus, the ADMM is finally given by the iterates

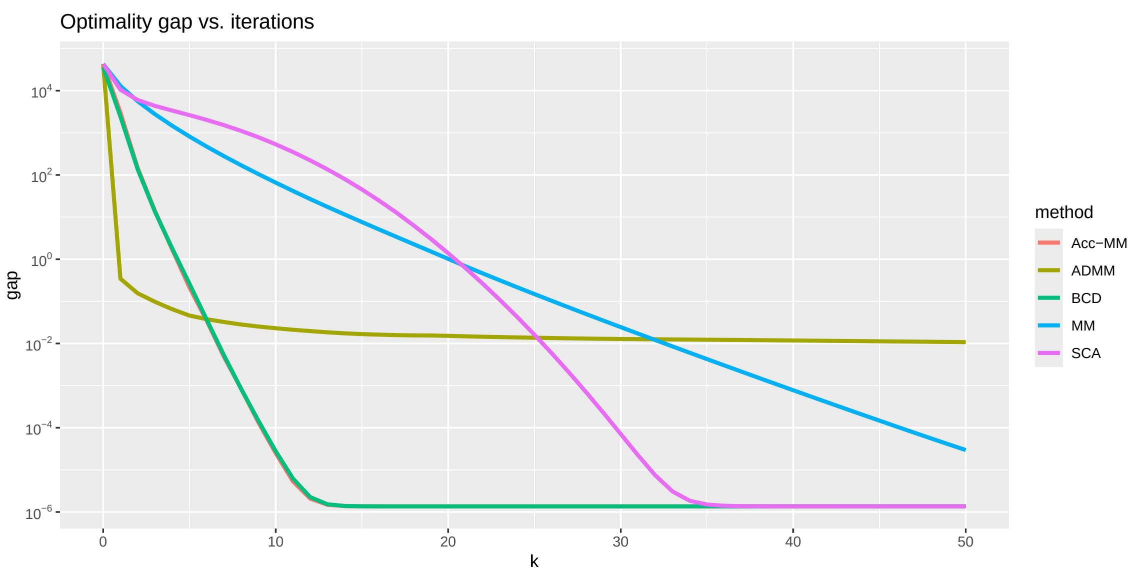
$$\left.\begin{aligned} \boldsymbol{x}^{k+1} &= \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho\left(\boldsymbol{z}^k - \boldsymbol{u}^k\right)\right) \\ \boldsymbol{z}^{k+1} &= \mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x}^{k+1} + \boldsymbol{u}^k\right) \\ \boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \left(\boldsymbol{x}^{k+1} - \boldsymbol{z}^{k+1}\right) \end{aligned}\right\} \qquad k = 0, 1, 2, \ldots$$

Convergence of ADMM for the ℓ_2–ℓ_1-norm minimization:

```r
# Set up
set.seed(42)
x0 <- rnorm(n)
soft_thresholding <- function(u, lmd) sign(u)*pmax(0, abs(u) - lmd)
num_times <- 10L  # to compute the cpu time


# Solve problem via ADMM
rho <- 1.0
x <- x0
z <- x
u <- rep(0, n)
cpu_time <- microbenchmark({
  Atb <- t(A) %*% b
  AtA <- t(A) %*% A
  }, unit = "microseconds", times = num_times)$time |> median()
df <- rbind(df, list(
  "k"           = 0,
  "cpu time k" = cpu_time,
  "CPU time [ms]" = NA,
  "gap"         = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
  "method"      = "ADMM")
  )
for (k in 1:50) {
  cpu_time <- microbenchmark({
    x_new <- solve(AtA + rho*diag(n), Atb + rho*(z - u))
    z_new <- soft_thresholding(x_new + u, lmd/rho)
    u_new <- u + (x_new - z_new)
    }, unit = "microseconds", times = num_times)$time |> median()
  x <- x_new
  z <- z_new
  u <- u_new

  df <- rbind(df, list(
    "k"           = k,
    "cpu time k" = cpu_time,
    "CPU time [ms]" = NA,
    "gap"         = 0.5*sum((A %*% x - b)^2) + lmd*sum(abs(x)) - opt_value,
    "method"      = "ADMM")
    )
}
```

Plot convergence:

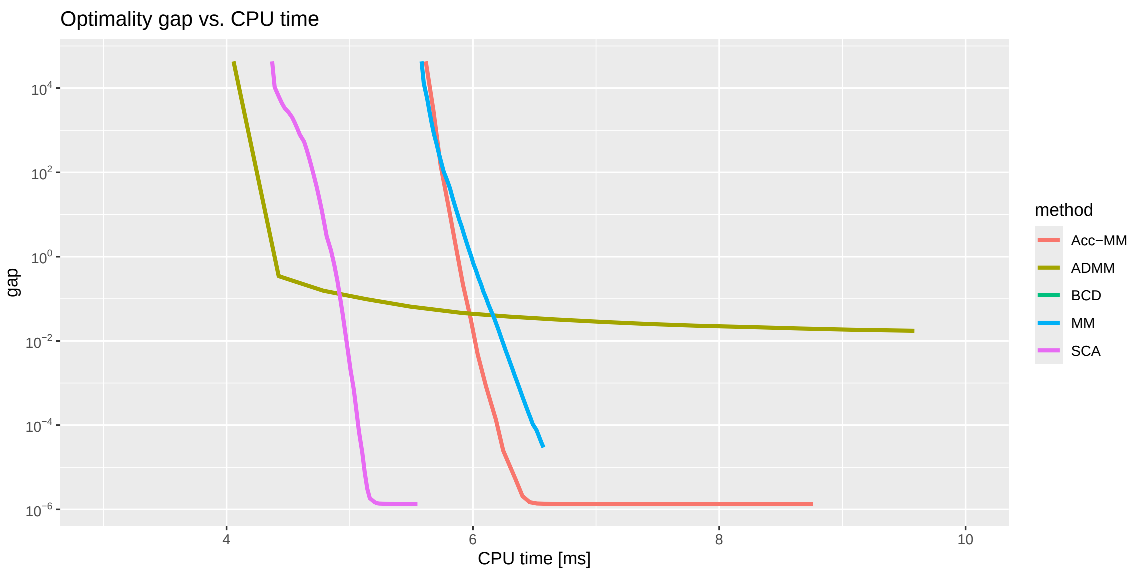
```r
library(ggplot2)
library(dplyr)
library(scales)

# Compute cumulative CPU time over iterations
df <- df |>
  group_by(method) |>
  mutate("CPU time [ms]" = cumsum(`cpu time k`)/1e6) |>
  ungroup()

# Plots
df |>
  ggplot(aes(x = k, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  ggtitle("Optimality gap vs. iterations")
```




```r
df |>
  ggplot(aes(x = `CPU time [ms]`, y = gap, color = method)) +
  geom_line(linewidth = 1.2) +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels = trans_format("log10", math_format(10^.x))) +
  xlim(c(3, 10)) +
  ggtitle("Optimality gap vs. CPU time")
```

```
Warning: Removed 87 rows containing missing values or values outside the scale range
```

(`geom_line()`).


Optimality gap vs. CPU time
gap
10^4
10^2
10^0
10^{-2}
10^{-4}
10^{-6}
4
6
8
10
CPU time [ms]
method
Acc−MM
ADMM
BCD
MM
SCA