Solutions to Exercises

Portfolio Optimization: Theory and Application
Chapter 11 – Risk Parity Portfolios

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

portfoliooptimizationbook.com

Exercise 11.1: Change of variable

Show why $\boldsymbol{\Sigma x} = \boldsymbol{b}/\boldsymbol{x}$ can be equivalently solved as $\boldsymbol{Cx} = \boldsymbol{b}/\boldsymbol{x}$, where $\boldsymbol{C}$ is the correlation matrix defined as $\boldsymbol{C} = \boldsymbol{D}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{D}^{-1/2}$ with $\boldsymbol{D}$ a diagonal matrix containing $\text{diag}(\boldsymbol{\Sigma})$ along the main diagonal. Would it be possible to use instead $\boldsymbol{C} = \boldsymbol{M}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{M}^{-1/2}$, where $\boldsymbol{M}$ is not necessarily a diagonal matrix?

Solution

Start by writing $\boldsymbol{\Sigma x} = \boldsymbol{b}/\boldsymbol{x}$ as

$$\boldsymbol{D}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{D}^{-1/2}\tilde{\boldsymbol{x}} = \boldsymbol{D}^{-1/2}\boldsymbol{b}/(\boldsymbol{D}^{-1/2}\tilde{\boldsymbol{x}}) = \boldsymbol{b}/\tilde{\boldsymbol{x}},$$

where $\tilde{\boldsymbol{x}} = \boldsymbol{D}^{1/2}\boldsymbol{x}$. This leads to

$$\boldsymbol{C}\tilde{\boldsymbol{x}} = \boldsymbol{b}/\tilde{\boldsymbol{x}},$$

and we can recover $\boldsymbol{x}$ as $\boldsymbol{x} = \boldsymbol{D}^{-1/2}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}/\boldsymbol{\sigma}$, where $\boldsymbol{\sigma}$ denote the volatilities, i.e., the diagonal elements of $\boldsymbol{D}^{1/2}$.

Now, to see if it would be possible to use instead $\boldsymbol{C} = \boldsymbol{M}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{M}^{-1/2}$, where $\boldsymbol{M}$ is not necessarily a diagonal matrix, let's proceed similarly:

$$\boldsymbol{M}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{M}^{-1/2}\tilde{\boldsymbol{x}} = \boldsymbol{M}^{-1/2}\boldsymbol{b}/(\boldsymbol{M}^{-1/2}\tilde{\boldsymbol{x}}),$$

where $\tilde{\boldsymbol{x}} = \boldsymbol{M}^{1/2}\boldsymbol{x}$. The issue here is that

$$\boldsymbol{M}^{-1/2}\boldsymbol{b}/(\boldsymbol{M}^{-1/2}\tilde{\boldsymbol{x}}) \neq \boldsymbol{b}/\tilde{\boldsymbol{x}}$$

because the matrix $\boldsymbol{M}$ is not diagonal, so it seems that a nondiagonal matrix cannot be used.

Exercise 11.2: Naive diagonal risk parity portfolio

If the covariance matrix is diagonal, $\boldsymbol{\Sigma} = \boldsymbol{D}$, then the system of nonlinear equations $\boldsymbol{\Sigma x} = \boldsymbol{b}/\boldsymbol{x}$ has the closed-form solution $\boldsymbol{x} = \sqrt{\boldsymbol{b}/\text{diag}(\boldsymbol{D})}$. Explore whether a closed-form solution can be obtained for the rank-one plus diagonal case $\boldsymbol{\Sigma} = \boldsymbol{u}\boldsymbol{u}^\mathsf{T} + \boldsymbol{D}$.

Solution

If $\boldsymbol{\Sigma} = \boldsymbol{D} = \text{Diag}(\boldsymbol{d})$, then $\boldsymbol{\Sigma x} = \boldsymbol{b}/\boldsymbol{x}$ can be written as

$$\boldsymbol{D x} = \text{Diag}(\boldsymbol{d})\boldsymbol{x} = \boldsymbol{d} \odot \boldsymbol{x} = \boldsymbol{b}/\boldsymbol{x}$$

which leads to

$$\boldsymbol{x}^2 = \boldsymbol{b}/\boldsymbol{d}$$

or

$$\boldsymbol{x} = \sqrt{\boldsymbol{b}/\boldsymbol{d}}.$$

Now, to see whether a closed-form solution can still be obtained for the rank-one plus diagonal case $\boldsymbol{\Sigma} = \boldsymbol{u}\boldsymbol{u}^\mathsf{T} + \boldsymbol{D}$, let's proceed similarly:

$$(\boldsymbol{u}\boldsymbol{u}^\mathsf{T} + \boldsymbol{D})\boldsymbol{x} = (\boldsymbol{u}^\mathsf{T}\boldsymbol{x})\boldsymbol{u} + \boldsymbol{d} \odot \boldsymbol{x} = \boldsymbol{b}/\boldsymbol{x}.$$

This leads to

$$(\boldsymbol{u}^\mathsf{T}\boldsymbol{x})(\boldsymbol{u}/\boldsymbol{d}) \odot \boldsymbol{x} + \boldsymbol{x}^2 = \boldsymbol{b}/\boldsymbol{d},$$

which does not seem to simplify as before. However, this can still be solved with the closed-form solution to a second-order equation:

$$\boldsymbol{x}^2 + (\boldsymbol{u}^\mathsf{T}\boldsymbol{x})(\boldsymbol{u}/\boldsymbol{d}) \odot \boldsymbol{x} - \boldsymbol{b}/\boldsymbol{d} = \boldsymbol{0}$$

with positive solution given by

$$x_i = \frac{-(\boldsymbol{u}^\mathsf{T}\boldsymbol{x})(u_i/d_i) + \sqrt{((\boldsymbol{u}^\mathsf{T}\boldsymbol{x})(u_i/d_i))^2 + 4b_i/d_i}}{2}, \qquad i = 1, \cdots, n$$

or, in vector form,

$$\boldsymbol{x} = \frac{-(\boldsymbol{u}^\mathsf{T}\boldsymbol{x})(\boldsymbol{u}/\boldsymbol{d}) + \sqrt{((\boldsymbol{u}^\mathsf{T}\boldsymbol{x})(\boldsymbol{u}/\boldsymbol{d}))^2 + 4(\boldsymbol{b}/\boldsymbol{d})}}{2}.$$

Exercise 11.3: Vanilla convex risk parity portfolio

The solution to the formulation

$$\begin{array}{ll} \underset{\boldsymbol{x}\geq\mathbf{0}}{\text{maximize}} & \boldsymbol{b}^\mathsf{T}\log(\boldsymbol{x}) \\ \text{subject to} & \sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}} \leq \sigma_0 \end{array}$$

is

$$\lambda\boldsymbol{\Sigma}\boldsymbol{x} = \boldsymbol{b}/\boldsymbol{x} \times \sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}}.$$

Can you solve for λ and rewrite the solution in a more compact way without λ?

Solution

We can left-multiply both sides of $\lambda\boldsymbol{\Sigma}\boldsymbol{x}/\sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}} = \boldsymbol{b}/\boldsymbol{x}$ by $\boldsymbol{x}^\mathsf{T}$ to obtain:

$$\lambda\sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}} = \boldsymbol{x}^\mathsf{T}(\boldsymbol{b}/\boldsymbol{x}) = \mathbf{1}^\mathsf{T}\boldsymbol{b}.$$

Noting that at an optimal point it must be that the constraint is satisfied with equality: $\sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}} = \sigma_0$ (otherwise the objective value could be further increased), we can simplify it to

$$\lambda\sigma_0 = \mathbf{1}^\mathsf{T}\boldsymbol{b},$$

which leads to $\lambda = \mathbf{1}^\mathsf{T}\boldsymbol{b}/\sigma_0$. Then we can finally write the solution as

$$\frac{\mathbf{1}^\mathsf{T}\boldsymbol{b}}{\sigma_0^2} \times \boldsymbol{\Sigma}\boldsymbol{x} = \boldsymbol{b}/\boldsymbol{x}.$$

Exercise 11.4: Newton's method

Newton's method requires computing the direction $\boldsymbol{d} = \mathsf{H}^{-1}\nabla f$ or, equivalently, solving the system of linear equations $\mathsf{H}\,\boldsymbol{d} = \nabla f$ for $\boldsymbol{d}$. Explore whether a more efficient solution is possible by exploiting the structure of the gradient and Hessian:

$$\begin{aligned} \nabla f &= \boldsymbol{\Sigma}\boldsymbol{x} - \boldsymbol{b}/\boldsymbol{x}, \\ \mathsf{H} &= \boldsymbol{\Sigma} + \mathrm{Diag}(\boldsymbol{b}/\boldsymbol{x}^2). \end{aligned}$$

Solution

Solving the system of linear equations $\mathsf{H}\,\boldsymbol{d} = \nabla f$ for $\boldsymbol{d}$ has a computational cost of $O(n^3)$. We will now assume $\boldsymbol{\Sigma}^{-1}$ has been precomputed and then the resolution will have a cost of $O(n^2)$. The key is to apply the matrix inversion lemma to the Hessian matrix $\mathsf{H} = \boldsymbol{\Sigma} + \boldsymbol{D}$, where $\boldsymbol{D} = \mathrm{Diag}(\boldsymbol{b}/\boldsymbol{x}^2)$,

as follows:

$$\mathsf{H}^{-1} = (\boldsymbol{\Sigma} + \boldsymbol{D})^{-1} = \boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\big(\boldsymbol{I} + \boldsymbol{D}\boldsymbol{\Sigma}^{-1}\big)^{-1}\boldsymbol{D}\boldsymbol{\Sigma}^{-1}.$$

In particular, the steps are the following:

1. Compute preliminary term

$$\boldsymbol{v} = \boldsymbol{\Sigma}^{-1}\nabla f = \boldsymbol{\Sigma}^{-1}\big(\boldsymbol{\Sigma}\boldsymbol{x} - \boldsymbol{b}/\boldsymbol{x}\big) = \boldsymbol{x} - \boldsymbol{\Sigma}^{-1}(\boldsymbol{b}/\boldsymbol{x}).$$

2. Form the matrix

$$\boldsymbol{M} = \boldsymbol{I} + \boldsymbol{D}\boldsymbol{\Sigma}^{-1}.$$

3. Solve for $\boldsymbol{z}$:

$$\boldsymbol{M}\boldsymbol{z} = \boldsymbol{D}\boldsymbol{v}.$$

4. Compute final direction

$$\boldsymbol{d} = \boldsymbol{v} - \boldsymbol{\Sigma}^{-1}\boldsymbol{z}.$$

Exercise 11.5: MM algorithm

The MM algorithm requires the computation of the largest eigenvalue $\lambda_{\max}$ of matrix $\boldsymbol{\Sigma}$, which can be obtained from the eigenvalue decomposition of the matrix. A more efficient alternative is the *power iteration method*. Program both methods and compare their computational complexity.

Solution

First, we generate the covariance matrix:

```
library(microbenchmark)

# Generate covariance matrix
set.seed(42)
n <- 100
A <- matrix(rnorm(n^2), n, n)
Sigma <- t(A) %*% A
```

Then, we compute the maximum eigenvalue with the built-in function `eigen()`:

```r
direct_nanoseconds = microbenchmark({
  lmd_max <- max(eigen(Sigma)$values)
  }, unit = "nanoseconds", times = 100L)$time |> median()

cat(direct_nanoseconds, "nanoseconds used by the built-in function eigen()
    to compute the maximum eigenvalue of", lmd_max)
```

```
2743492 nanoseconds used by the built-in function eigen()
    to compute the maximum eigenvalue of 377.2282
```

Finally, we employ 20 iterations of the power iteration method (the number of iterations depends on the accuracy desired):

```r
x0 <- rnorm(n)
power_iteration_nanoseconds = microbenchmark({
  u <- x0; for (i in 1:20) u <- Sigma %*% u
  lmd_max <- as.numeric(t(u) %*% Sigma %*% u / sum(u^2))
  }, unit = "nanoseconds", times = 100L)$time |> median()

cat(power_iteration_nanoseconds, "nanoseconds used by the power iteration method
    to compute the maximum eigenvalue of", lmd_max)
```

```
1675598 nanoseconds used by the power iteration method
    to compute the maximum eigenvalue of 375.2165
```

Exercise 11.6: Coordinate descent vs. SCA methods

Consider the vanilla convex formulation

$$\underset{\boldsymbol{x}\geq\mathbf{0}}{\text{minimize}} \quad \tfrac{1}{2}\boldsymbol{x}^{\mathsf{T}}\boldsymbol{\Sigma}\boldsymbol{x} - \boldsymbol{b}^{\mathsf{T}}\log(\boldsymbol{x}).$$

Implement the cyclical coordinate descent method and the parallel SCA method in a high-level programming language (e.g., R, Python, Julia, or MATLAB) and compare the convergence against the CPU time for these two methods. Then, re-implement these two methods in a low-level programming language (e.g., C, C++, C#, or Rust) and compare the convergence again. Comment on the difference observed.

Solution

Let's construct a covariance matrix from stock market data:

```
library(xts)
library(pob)       # Market data used in the book
library(riskParityPortfolio)

# Prep data
N <- 200
Sigma <- cov(diff(log(SP500_2015to2020$stocks[, 1:N]))[-1])
sigma <- sqrt(diag(Sigma))
C <- cov2cor(Sigma)
b <- rep(1/N, N)
w_opt <- riskParityPortfolio(Sigma, b = b)$w
x_opt <- w_opt / as.vector(sqrt(w_opt %*% Sigma %*% w_opt))
opt_value <- 0.5 * x_opt %*% Sigma %*% x_opt - b %*% log(x_opt)

num_iter  <- 10L
num_times <- 10L  # to compute the cpu time
```

We can start with the cyclical coordinate descent method for the function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x} - \boldsymbol{b}^\mathsf{T}\log(\boldsymbol{x})$. The elementwise minimization becomes

$$\underset{x_i \geq 0}{\text{minimize}} \quad \tfrac{1}{2}x_i^2\boldsymbol{\Sigma}_{ii} + x_i(\boldsymbol{x}_{-i}^\mathsf{T}\boldsymbol{\Sigma}_{-i,i}) - b_i \log x_i,$$

where $\boldsymbol{x}_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)$ denotes the variable $\boldsymbol{x}$ without the ith element and $\boldsymbol{\Sigma}_{-i,i}$ denotes the ith column of matrix $\boldsymbol{\Sigma}$ without the ith element. Setting the partial derivative with respect to x_i to zero gives us the second-order equation

$$\boldsymbol{\Sigma}_{ii}x_i^2 + (\boldsymbol{x}_{-i}^\mathsf{T}\boldsymbol{\Sigma}_{-i,i})x_i - b_i = 0$$

with positive solution given by

$$x_i = \frac{-\boldsymbol{x}_{-i}^\mathsf{T}\boldsymbol{\Sigma}_{-i,i} + \sqrt{(\boldsymbol{x}_{-i}^\mathsf{T}\boldsymbol{\Sigma}_{-i,i})^2 + 4\boldsymbol{\Sigma}_{ii}b_i}}{2\boldsymbol{\Sigma}_{ii}}.$$

```
library(microbenchmark)
library(dplyr)

#
# Cyclical Spinu coordinate descent algorithm
#
x <- sqrt(b)/sqrt(rowSums(Sigma))
df <- data.frame(
  "k"          = 0L,
  "cpu time k" = 0,
  "obj_value"  = 0.5 * x %*% Sigma %*% x - b %*% log(x),
  "gap"        = 0.5 * x %*% Sigma %*% x - b %*% log(x) - opt_value,
  "method"     = "Cyclical coordinate descent",
  check.names = FALSE
  )

for (k in 1:num_iter) {
  cpu_time <- microbenchmark({
    x_new <- x
    for (i in 1:N) {
      Sigma_xk_i <- as.numeric(x_new[-i] %*% Sigma[-i, i])
      x_new[i] <- (- Sigma_xk_i + sqrt(Sigma_xk_i^2 + 4*Sigma[i, i]*b[i]))/(2*Sigma[i, i])
    }
    }, unit = "microseconds", times = num_times)$time |> median()
  x <- as.numeric(x_new)

  df <- rbind(df, list(
    "k"          = k,
    "cpu time k" = cpu_time,
    "obj_value"  = 0.5 * x %*% Sigma %*% x - b %*% log(x),
    "gap"        = 0.5 * x %*% Sigma %*% x - b %*% log(x) - opt_value,
    "method"     = "Cyclical coordinate descent"))
}
```

The SCA method obtains the iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ by solving a sequence of simpler surrogate problems. In particular, the following surrogate can be used for the term $\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}$ around the current point $\boldsymbol{x} = \boldsymbol{x}^k$:

$$\frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x} \approx \frac{1}{2}(\boldsymbol{x}^k)^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{x}^k + (\boldsymbol{\Sigma}\boldsymbol{x}^k)^\mathsf{T}(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^\mathsf{T}\mathrm{Diag}(\boldsymbol{\Sigma})(\boldsymbol{x} - \boldsymbol{x}^k),$$

where $\mathrm{Diag}(\boldsymbol{\Sigma})$ is a diagonal matrix containing the diagonal of $\boldsymbol{\Sigma}$. We can now solve our original problems by solving instead a sequence of surrogate problems

$$\underset{\boldsymbol{x}\geq\boldsymbol{0}}{\text{minimize}} \quad \tfrac{1}{2}\boldsymbol{x}^\mathsf{T}\mathrm{Diag}(\boldsymbol{\Sigma})\boldsymbol{x} + \boldsymbol{x}^\mathsf{T}(\boldsymbol{\Sigma} - \mathrm{Diag}(\boldsymbol{\Sigma}))\boldsymbol{x}^k - \boldsymbol{b}^\mathsf{T}\log(\boldsymbol{x}),$$

from which setting the gradient to zero gives the second-order equation

$$\boldsymbol{\Sigma}_{ii}x_i^2 + ((\boldsymbol{\Sigma} - \mathrm{Diag}(\boldsymbol{\Sigma}))\boldsymbol{x}^k)_i x_i - b_i = 0$$

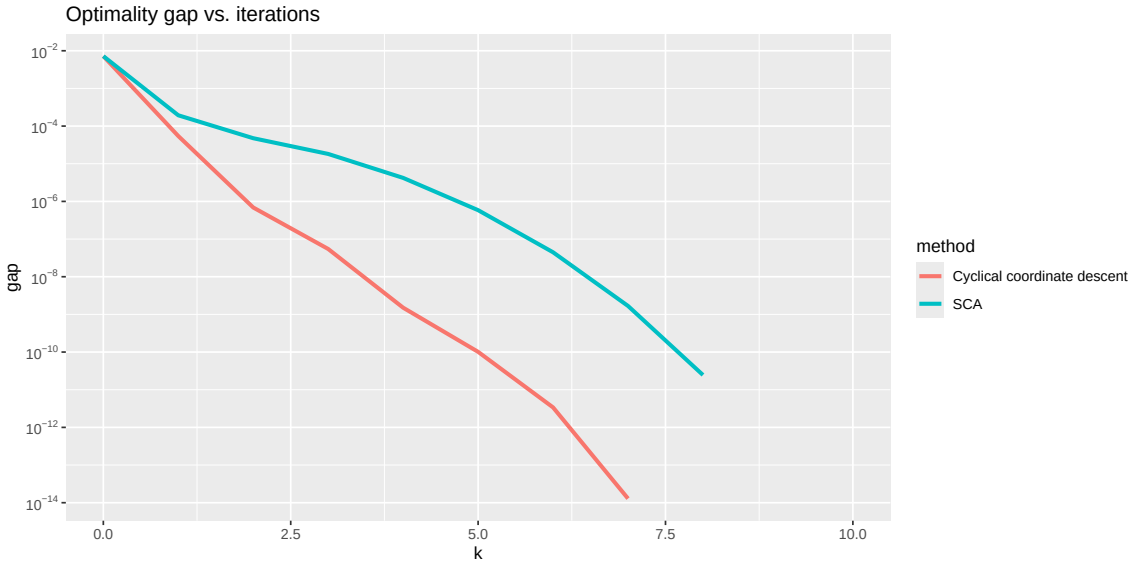
with positive solution

$$x_i = \frac{-((\boldsymbol{\Sigma} - \mathrm{Diag}(\boldsymbol{\Sigma}))\boldsymbol{x}^k)_i + \sqrt{((\boldsymbol{\Sigma} - \mathrm{Diag}(\boldsymbol{\Sigma}))\boldsymbol{x}^k)_i^2 + 4\boldsymbol{\Sigma}_{ii} b_i}}{2\boldsymbol{\Sigma}_{ii}}.$$

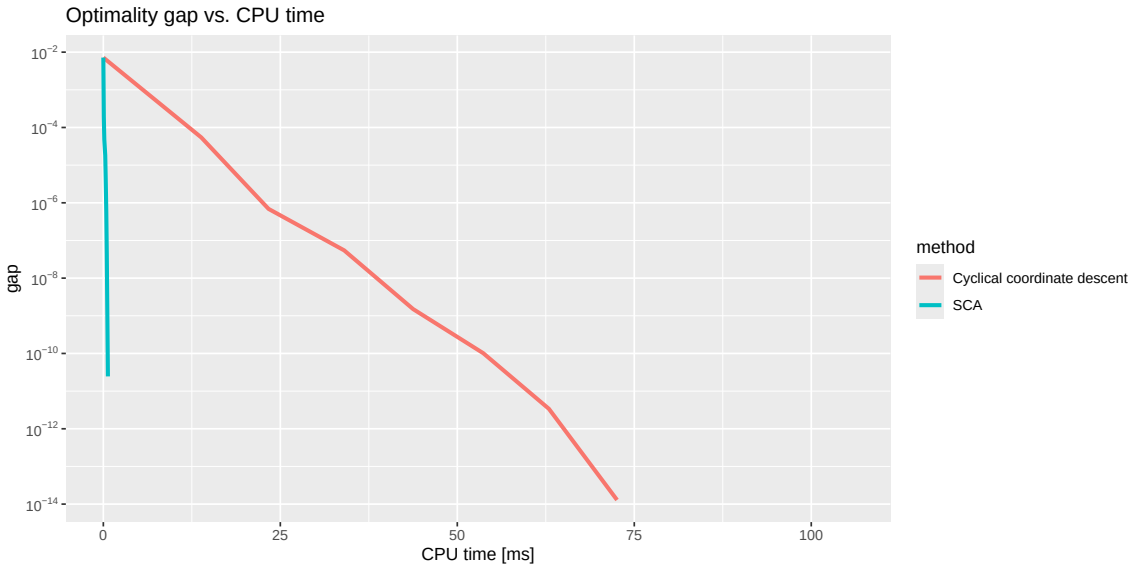
```
#
# Parallel Spinu SCA
#
x <- sqrt(b)/sqrt(rowSums(Sigma))
gamma <- 1
eps <- 0.1
df <- rbind(df, list(
  "k"          = 0L,
  "cpu time k" = 0,
  "obj_value"  = 0.5 * x %*% Sigma %*% x - b %*% log(x),
  "gap"        = 0.5 * x %*% Sigma %*% x - b %*% log(x) - opt_value,
  "method"     = "SCA")
  )

Sigma_Diag_Sigma <- Sigma - diag(diag(Sigma))
for (k in 1:num_iter) {
  cpu_time <- microbenchmark({
    Sigma_Diag_Sigma_xk <- Sigma_Diag_Sigma %*% x
    x_hat <- (-Sigma_Diag_Sigma_xk + sqrt(Sigma_Diag_Sigma_xk^2 + 4*diag(Sigma)*b))/(2*diag(Sigma))
    x_new <- gamma*x_hat + (1 - gamma)*x
    }, unit = "microseconds", times = num_times)$time |> median()
  x <- as.numeric(x_new)
  gamma <- gamma * (1 - eps*gamma)

  df <- rbind(df, list(
    "k"          = k,
    "cpu time k" = cpu_time,
    "obj_value"  = 0.5 * x %*% Sigma %*% x - b %*% log(x),
    "gap"        = 0.5 * x %*% Sigma %*% x - b %*% log(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))) +
  ggtitle("Optimality gap vs. CPU time")
```



We can observe the much faster convergence of the parallel SCA method compared to the sequential cyclical coordinate descent.

We leave the implementation in a low-level programming language (e.g., C, C++, C#, or Rust) to the user to observe whether the difference in convergence speed reduces or remains the same.