Nelder Mead optimization

Yanfei Kang

Import libraries:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(animation)
```

Write my own NelderMead function:

```
NelderMead <- function(x, fun, tol = 1e-08, alpha = 1, gamma = 2, lo = 0.5, sig = 0.5,
    maxiter = 50) {
    n <- dim(x)[2]  # The number of dimensions.
    num <- 0
    df <- fun(x) %>%
        cbind(x, num)
    names(df)[c(1, n + 2)] = c("f", "num")
    area <- tol + 1
    trc <- df[FALSE, ]

    while (area > tol & num < maxiter) {
        df[n + 2] <- num <- num + 1

        # Sort the f(x).
        df <- df %>%
            arrange(f)
        f1 <- df[1, 1]
        fn <- df[n, 1]
        xw <- df[n + 1, 2:(n + 1)]
        x0 <- colMeans(df[1:n, 2:(n + 1)])
        trc <- rbind(trc, df)

        # Compute reflected point.
        xr <- x0 + alpha * (x0 - xw)
        fr <- fun(xr)

        if (f1 <= fr & fr < fn) {
```

```
            # Keep xr.
            df[n + 1, 1:(n + 1)] <- cbind(fr, xr)
        } else if (fr < f1) {
            # Compute reflected point.
            xe <- x0 + gamma * (xr - x0)
            fe <- fun(xe)

            if (fe < fr) {
                # Keep xe.
                df[n + 1, 1:(n + 1)] <- cbind(fe, xe)
            } else {
                # Keep xr.
                df[n + 1, 1:(n + 1)] <- cbind(fr, xr)
            }
        } else {
            # Compute contracted point.
            xc <- x0 + lo * (xw - x0)
            fc <- fun(xc)

            if (fc < df[n + 1, 1]) {
                # Keep xc.
                df[n + 1, 1:(n + 1)] <- cbind(fc, xc)
            } else {
                # Shrink.
                x1 <- t(matrix(df[1, 2:(n + 1)]))
                df[, 2:(n + 1)] <- x <- x1 + sig * (df[, 2:(n + 1)] - x1)
                df[, 1] <- fun(x)
            }
        }

        area <- cbind(df[, 2:(n + 1)], 1) %>%
            as.matrix %>%
            det %>%
            abs/2

        # print(df) cat(area, end = '\n')
    }
    df[n + 2] <- num + 1
    trc <- rbind(trc, df)

    if (num == maxiter) {
        cat("\nMaximum number of iterations was reached!")
    }

    df0 = colMeans(df)
    return(list(x = df[2:(n + 1)], f = df[1], f.x = df0, trace = trc))
}
```

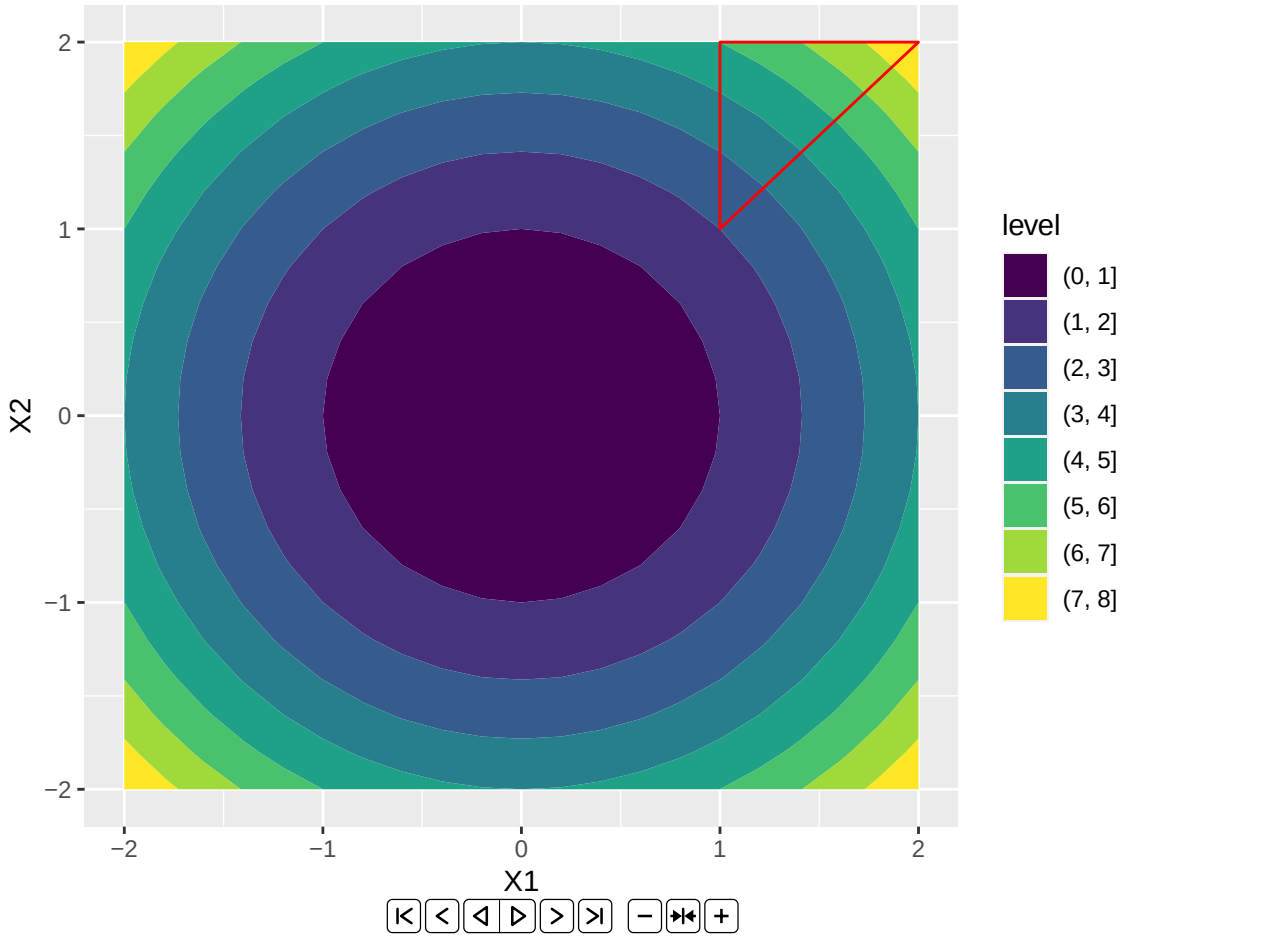
Use a triangle with vertices (1,1),(1,2),(2,2) as the starting simplex to find the minimum of the function $f(x) = x_1^2 + x_2^2$.

```
f <- function(x) x[1]^2 + x[2]^2
v1 <- c(1, 1)
v2 <- c(1, 2)
```

```
v3 <- c(2, 2)
v <- rbind(v1, v2, v3) %>%
    data.frame

r <- NelderMead(v, f, maxiter = 80)
```

Display the trace of iterations in GIF on a filled contour plot.

```
tr <- r$trace
n <- tr["num"] %>%
    max  # Iteration times.

x <- seq(-2, 2, 0.2) %>%
    as.data.frame
y <- seq(-2, 2, 0.2) %>%
    as.data.frame
ff <- merge(x, y, by = NULL)
ff <- mutate(ff, z = ..x^2 + ..y^2)
for (i in 1:n) {
    p <- ggplot(tr[(3 * i - 2):(3 * i), ], aes(X1, X2)) + geom_contour_filled(aes(x = ..x,
        y = ..y, z = z), ff) + geom_polygon(fill = NA, colour = "red")

    plot(p)
}
```



References

- R Markdown Cookbook: https://bookdown.org/yihui/rmarkdown-cookbook/animation.html.