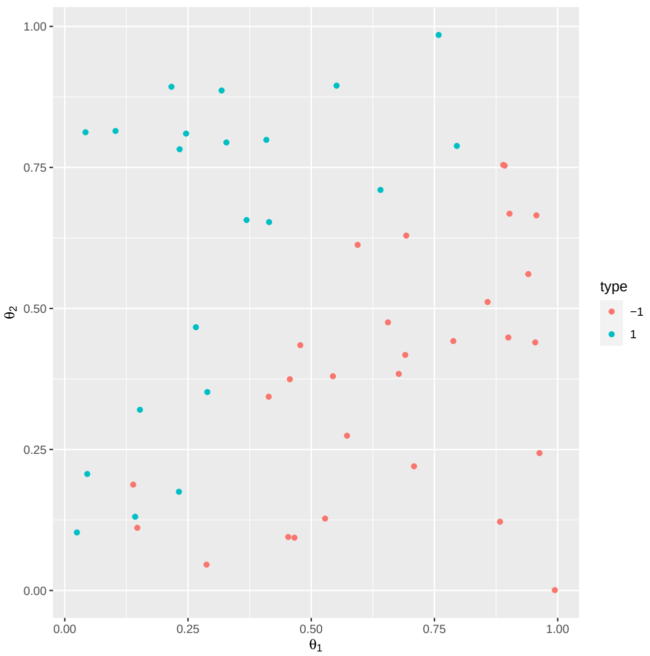
Univariate Optimization 1

Exercise 1: Golden Ratio, Brent's Method

```
library(ggplot2)

set.seed(123)

X = matrix(runif(100), ncol = 2)
y = -((X %*% c(-1, 1) + rnorm(50, 0, 0.1) < 0) * 2 - 1)
df = as.data.frame(X)
df$type = as.character(y)

ggplot(df) +
  geom_point(aes(x = V1, y = V2, color=type)) +
  xlab(expression(theta[1])) +
  ylab(expression(theta[2]))
```



(a) Since f is convex it holds for arbitrary $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2, t \in [0,1]$ that
$f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) \leq f(\mathbf{x}) + t(f(\mathbf{y}) - f(\mathbf{x}))$.
This means this holds also for $\mathbf{x}_c = (x, c)^\top$ and $\mathbf{y}_c = (y, c)^\top$ with $x, y \in \mathbb{R}$ and fixed $c \in \mathbb{R}$:
$f(\mathbf{x}_c + t(\mathbf{y}_c - \mathbf{x}_c)) \leq f(\mathbf{x}_c) + t(f(\mathbf{y}_c) - f(\mathbf{x}_c)) \iff g_c(x + t(y - x)) \leq g_c(x) + t(g_c(y) - g_c(x))$.
$\Rightarrow g_c$ is convex.

(b) The non-geometric primal linear SVM formulation is convex and unconstrained $\Rightarrow$ For one parameter the objective is also convex (a)) and we can directly use GR . In contrast, the geometric formulation has linear constraints.

(c)
```r
# Define objective
f <- function(theta) theta %*% theta +
    sum(sapply(1 - y * (X %*% theta), function(x) max(x, 0)))

# Objective w.r.t theta_1 with fixed theta_2
ft1 <- function(theta_1) f(c(theta_1, 2))

phi = (sqrt(5) - 1)/2

gr <- function(f, lx=-3, rx=3, abs_error = 0.01){

  # initialize variables needed for stopping criterion
  fbest_old = Inf
  xbest_old = Inf
  xbest = NA

  # compute candidate xs
  dist = rx - lx
  cx = c(lx + (1-phi) * dist, rx - (1-phi) * dist)

  while(TRUE){
    fcx1 = f(cx[1])
    fcx2 = f(cx[2])

    # check which candidate is better and update cx
    if (fcx1 < fcx2){
      fbest = fcx1
      xbest = cx[1]
      rx = cx[2]
      cx[2] = cx[2] - (cx[1] - lx)
    }else{
      fbest = fcx2
      xbest = cx[2]
      lx = cx[1]
      cx[1] = cx[1] + (rx - cx[2])
    }
    # assure cx[1] < cx[2]
    cx = sort(cx)

    # check if we need to stop the loop depending on the termination criterion
    if (abs(xbest_old - xbest) < abs_error){
      return(c(xbest, fbest))
    }
    fbest_old = fbest
    xbest_old = xbest
  }
}

gr(ft1)

## [1] -2.45898 29.91294
```

(d) We are given three equations:
$ax_1^2 + bx_1 + c = y_1$
$ax_2^2 + bx_2 + c = y_2$

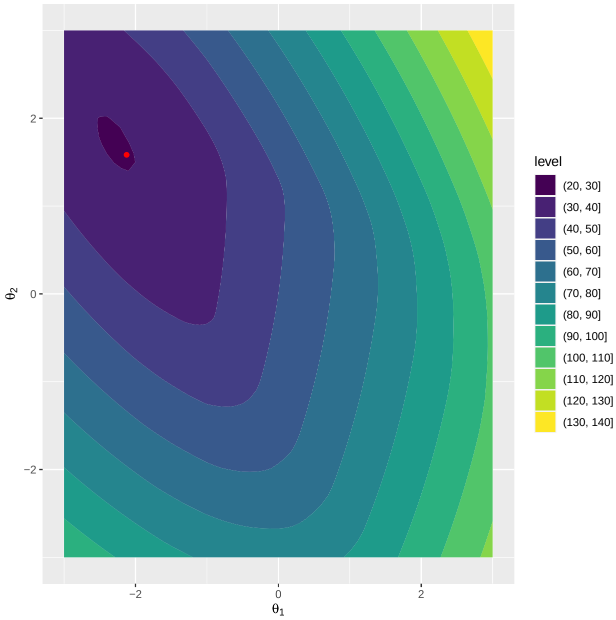
$ax_3^2 + bx_3 + c = y_3$. Which we can express equivalently as $\underbrace{\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}}_{:=\boldsymbol{\Lambda}} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$. The result follows straightforwardly assuming $\boldsymbol{\Lambda}$ is non-singular.

(e)
```
gr_step <- function(f, lx, rx){
  dist = rx - lx

  # compute candidates
  cxs = c(lx + (1-phi) * dist,
    rx - (1-phi) * dist)

  fcx = sapply(cxs, f)

  # find best candidate
  if (fcx[1] < fcx[2]){
    fbest = fcx[1]
    cx = cxs[1]
    rx = cxs[2]
  }else{
    fbest = fcx[2]
    cx = cxs[2]
    lx = cxs[1]
  }

  return(c(lx, rx, cx, fbest))
}

brent <- function(f, lx = -3, rx = 3, abs_error = 0.01){

  fbest_old = Inf
  xbest = NA
  xbest_old = Inf

  # we do not have a valid candidate in the beginning
  cx = Inf

  while(TRUE){
    # if candidate is not valid do a golden ratio step
    if(cx <= lx | cx >= rx){
      res = gr_step(f, lx, rx)
      lx = res[1]
      rx = res[2]
      cx = res[3]
      xbest = cx
      fbest = res[4]
    }else{ # try doing quadratic interpolation otherwise
      # compute objective values
      xs = c(lx, rx, cx)
      fxs = sapply(xs, ft1)

      # find parameters of the interpolating parabola
      params = solve(t(sapply(xs, function(x) c(x^2, x, 1))), fxs)
      # find minimum of the parabola
      cx_new = -params[2]/(2*params[1])

      # if candidate is valid do quadratic interpolation step
      if(cx_new < rx & cx_new > lx){
```

```r
        cxs = sort(c(cx, cx_new))
        fcx = sapply(cxs, f)

        # find best candidate
        if (fcx[1] < fcx[2]){
          fbest = fcx[1]
          cx = cxs[1]
          rx = cxs[2]
        }else{
          fbest = fcx[2]
          cx = cxs[2]
          lx = cxs[1]
        }
        xbest = cx

      }
    }

  # check if we need to stop the loop depending on the termination criterion
  if (abs(xbest - xbest_old) < abs_error){
    return(c(xbest, fbest))
  }
  fbest_old = fbest
  xbest_old = xbest
  }
}

brent(ft1)

## [1] -2.409456 29.903281
```

(f)
```r
# intialize thetas
t1 = 0
t2 = 0

for(i in 0:9){
  # alternate between univariately optimizing each parameter while the other
  # is fixed
  if(i %% 2 == 0){
    ft <- function(t) f(c(t, t2))
    res = gr(ft)
    t1 = res[1]

  }else{
    ft <- function(t) f(c(t1, t))
    res = gr(ft)
    t2 = res[1]
  }
  print(c(t1, t2, f(c(t1, t2))))
}

## [1] -1.583592  0.000000 37.878640
## [1] -1.583592  1.583592 32.490253
## [1] -2.124612  1.583592 29.742862
## [1] -2.124612  1.583592 29.742862
## [1] -2.124612  1.583592 29.742862
## [1] -2.124612  1.583592 29.742862
```

```
## [1] -2.124612  1.583592 29.742862
## [1] -2.124612  1.583592 29.742862
## [1] -2.124612  1.583592 29.742862
## [1] -2.124612  1.583592 29.742862

x = seq(-3, 3, by=0.1)
xx = expand.grid(X1 = x, X2 = x)

fxx = apply(xx, 1, f)
df = data.frame(xx = xx, fxx = fxx)

ggplot() +
    geom_contour_filled(data = df, aes(x = xx.X1, y = xx.X2, z = fxx)) +
    xlab(expression(theta[1])) +
    ylab(expression(theta[2])) +
    geom_point(data = as.data.frame(t(c(t1, t2))), mapping = aes(x=V1, y=V2),
               color="red")
```



(g) The trace looks like a orthogonal zig-zag line.

(h) In theory, yes, since our objective is convex and hence the objective w.r.t. each parameter is convex $\Rightarrow$ each step makes our objective as small as possible in this direction conditioned on the fixed parameters $\Rightarrow$ the found objective values are monotonically decreasing. However, depending on the curvature of the objective function this can take a long time because we can only "walk" in the direction of the coordinate axes (g)).