Penalty Method

$$\min f(x) = x_1^2 + x_2^2$$
$$\text{s.t.}$$
$$h(x) = \frac{1}{2}(x_1 - 3)^2 + x_2^2 - 5 = 0$$

$$g(x) = \frac{1}{2}x_1 + 1 - x_2 \leq 0$$

Rozwiązanie analityczne:
$L = x^2 + y^2 - \lambda_1(0.5x + 1 - y) - \lambda_2(0.5x^2 - 3x + 4.5 + y^2 - 5)$

Warunek 1. $\frac{\partial L}{\partial x} = 2x - 0.5\lambda_1 - \lambda_2 x + 3\lambda_2 = 0$
Warunek 2 . $\frac{\partial L}{\partial y} = 2y + \lambda_1 - 2\lambda_2 y = 0$
Warunek 3. $\frac{1}{2}(x-3)^2 + y^2 - 5 = 0$

Przypadek 1. $g(x)$ jest wiążące. Wtedy $0.5x + 1 - y = 0$ i $\lambda_1 > 0$
(...)
$3x^2 - 8x + 2 = 0$
(...)
$x^* = \frac{8-\sqrt{40}}{6} = 0.27924$
$y^* = 1 + \frac{8-\sqrt{40}}{12} = 1.13962$

Wczytanie bibliotek, definicja funkcji

In [2]:

```
library(plotly)
library(matlab)

ObjFun <- function(x) x[1]^2 + x[2]^2;
g <- function(x) 1/2 * x[1] + 1 - x[2];
h <- function(x) 1/2 * (x[1]-3)^2 + (x[2])^2 - 5;
```

Wizualizacja problemu

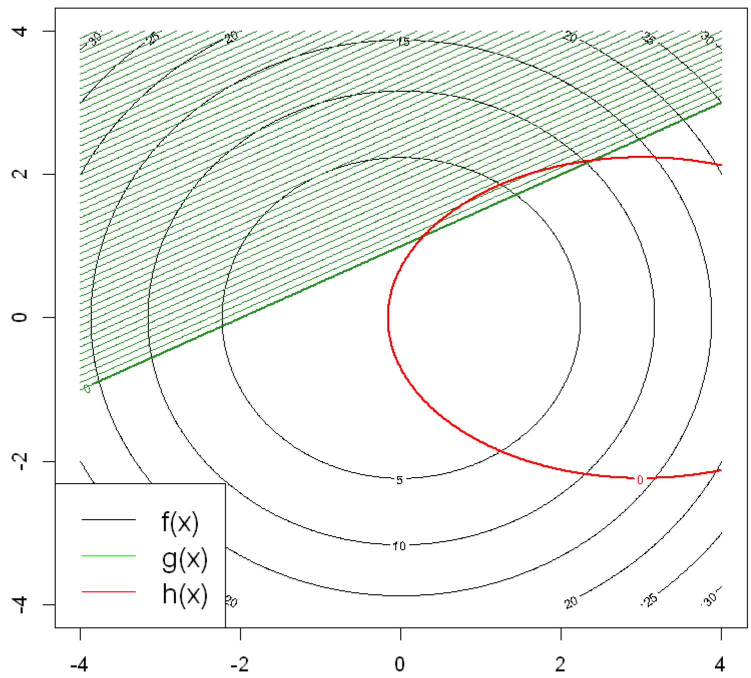
In [2]:

```
n_grid <- 500

f_val  <- matrix(NA, nrow =  n_grid, ncol = n_grid)
g_val  <- matrix(NA, nrow =  n_grid, ncol = n_grid)
h_val  <- matrix(NA, nrow =  n_grid, ncol = n_grid)

# Wypełnienie macierzy funkcji f, g, h
x_seq <- seq(-4, 4, length = n_grid)
for(i in 1 : n_grid){
    for(j in 1 : n_grid){
        ij_value=c(x_seq[i], x_seq[j])
        f_val[i,j] <- ObjFun(ij_value)
        g_val[i,j] <- g(ij_value)
        h_val[i,j] <- h(ij_value)
    }
}

# Wykresy warstwic
contour(x_seq, x_seq, f_val)
contour(x_seq, x_seq, g_val, levels = seq(0, -5, by = -0.1), col = 'forestgreen', add = T, drawlabels = F)
contour(x_seq, x_seq, g_val, levels = c(0), col = 'forestgreen', add = T, lwd = 2)
contour(x_seq, x_seq, h_val, levels = c(0), col = 'red', add = T, lwd = 2)
legend("bottomleft", legend=c("f(x)", "g(x)","h(x)"),col=c("black","green", "red"), lty=1:1:1, cex=1.4)
```



Funkcje pomocnicze

In [3]:

```
NumGrad <- function(f, x, h = 10^-6){
    n <- length(x)
    result <- rep(NA, n)
    for (i in 1 : n){
        e_i <- rep(0, n)
        e_i[i] <- 1
        result[i] <- (f(x+e_i*h) - f(x-e_i*h)) / (2*h)
    }
    return(result)
}
```

In [4]:

```
golden_section <- function(f, a, b, n = 20, e = NA){
    p <- (3-sqrt(5))/2 #0.3819

    if (a > b){ #zapewnia że a<b
        temp <- a
        a <- b
        b <- temp
    }

    if (!is.na(e)) n <- ceiling(log((b - a)/e) * (log(1 /(1-p)))^-1);

    d <- a+(b-a)*(1-p)
    yd <- f(d)

    for(i in 1 : n-1){
        c <- a+(b-a)*p
        yc <- f(c)

        if (yc < yd){
            b <- d
            d <- c
            yd <- yc

        }else{
            a <- b
            b <- c
        }
    }
    return(sort(c(a,b)))
}
```

In [5]:

```
SteepestDescent <- function(f, x, max_iter){
    n <- length(x)
    out <- list(x_hist = matrix(NA, nrow = max_iter, ncol = n),
                f_hist = rep(NA, max_iter),
                x_opt = x,
                f_opt = f(x))
    out$x_hist[1,] <- x
    out$f_hist[1] <- f(x)

    for(i in 2 : max_iter){
        grad <- NumGrad(f, x)

        param <- function(f, x_0, x_1){
            out <- list(f = function(a){
                    return(f((1-a) * x_0 + a*x_1))
                    })

            return(out)
        }

        a_k <- golden_section(param(f, x, x - grad)$f, 0, 2, e = 10^-4)

        x <- x - mean(a_k) * grad

        if (f(x)<out$f_opt){
            out$x_opt <- x
            out$f_opt <- f(x)
        }
        out$x_hist[i, ] <- x
        out$f_hist[i] <- f(x)

    }
    return(out)
}
```

Rozbicie funkcji kary $p_k(x)$:

$$\underbrace{p_k(x)}_{\text{funkcja kary}} = \underbrace{\gamma_k}_{\substack{\text{mnożnik} \\ \text{kary}}} \times \underbrace{p(x)}_{\substack{\text{k. bazowa} \\ \text{k. zasadnicza}}}$$

Funkcja kary zasadniczej

Dla ograniczeń nierównościowych, tzn. $g(x) \leq 0$:

- $p(x) = |\max(0, g(x)|$ - w praktyce nie stosuje się w solverach (nie różniczkowalna),
- $p(x) = (\max(0, g(x))^2$ - kara kwadratowa (różniczkowalna).

Dla ograniczeń równościowych, tzn. $h(x) = 0$:

- $p(x) = |h(x)|$ - podobnie, nie stosuje się w solverach (nie różniczkowalna),
- $p(x) = (h(x))^2$ - kara kwadratowa (różniczkowalna).

In [6]:

```
p_fun <- function(g = NA, h = NA){
    out_2 <- list(

        f = function(x) {
            out <- 0
            if (!is.na(g)){
                    for (g_i in g){
                        out <- out + max(0,g_i(x))^2
                    }
                }

            if (!is.na(h)){
                    for (h_i in h){
                        out <- out + h_i(x)^2
                    }
                }
            return(out)
            })
    return(out_2)
}
```

Implementacja metody funkcji kary

Mnożnik kary:

- $\forall_k \gamma_k > 0$
- $\forall_k \gamma_{k+1} > \gamma_k$
- $\lim_{k \to \infty} \gamma_k = \infty$

Mnożnik kary ustalany jest arbitralnie, może przyjąć postać np.: $\gamma_k = (1+\xi)^k, \xi > 0$
Funkcja pomocnicza przyjmuje postać:

$$f_k(x) = f(x) + \gamma^k \left[\sum_{i=1}^{m} h_i(x)^2 + \sum_{j=1}^{q} \max\left(g_q(x), 0\right)^2 \right]$$

Zwróćmy uwagę, że do rozwiązania problemu funkcji $f_k(x)$ potrzebujemy wykorzystać również solwer numeryczny!

In [7]:

```
penalty_method <- function(f, x, p, max_iter, e = 0.5){
    n <- length(x)

    ### Definicja funkcji pomocniczej 1###
        f_k <- function(x){
            return(f(x) + ((1+e)^i)*p(x))
        }

    out <- list(x_hist = matrix(NA, nrow = max_iter, ncol = n),
           f_hist = rep(NA, max_iter),
           x_opt = x,
           f_opt = f(x))

    out$x_hist[1,] <- x
    out$f_hist[1] <- f(x)

    for (i in 2 : max_iter){

        # Solwer optymalizacyjny
        sd <- SteepestDescent(f_k, x, 30)
        x <- sd$x_opt

        if (f(x)<out$f_opt){
            out$x_opt <- x
            out$f_opt <- f(x)
        }
        out$x_hist[i, ] <- x
        out$f_hist[i] <- f(x)
        out$multi[i] <- (1+e)^i
        out$p[i] <- p(x)
        out$f_k[i] <- f_k(x)

    }
    out$x_opt <- x
    out$f_opt <- f(x) + ((1+e)^max_iter)*p(x)
    return(out)
}
```

Wykonanie funkcji

In [8]:

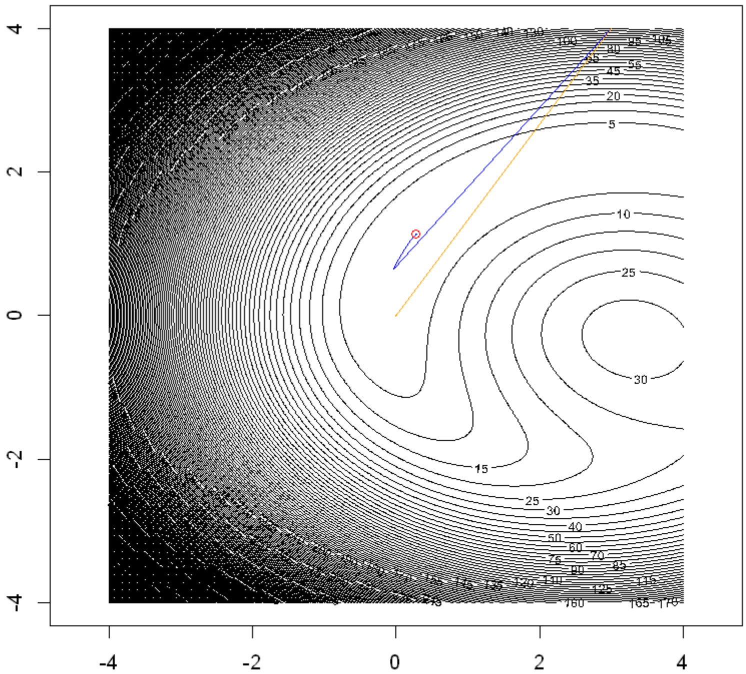
```
x0 <- c(3,4)
max_iter <- 20
pm <- penalty_method(ObjFun, x0, p = p_fun(c(g), c(h))$f, max_iter)
sd <- SteepestDescent(ObjFun, x0, max_iter)
```

Wizualizacje

In [9]:

```
p_val <- matrix(NA, nrow =  n_grid, ncol = n_grid)
p <- p_fun(c(g), c(h))$f
for(i in 1 : n_grid){
    for(j in 1 : n_grid){
        p_val[i,j] <- p(c(x_seq[i], x_seq[j]))
    }
}

contour(x_seq, x_seq, p_val, nlevels = 200, asp = T)
lines(pm$x_hist, type = 'l', col = 'blue')
lines(sd$x_hist, type = 'l', col = 'orange')
points(pm$x_opt[1],pm$x_opt[2],col='red')
```



In [10]:

```
graph_values <- matrix(NA, nrow =  n_grid, ncol = n_grid)
rho_graph<-(1.5)^max_iter
#rho_graph<-1
p <- p_fun(c(g), c(h))$f
for(i in 1 : n_grid){
    for(j in 1 : n_grid){
        graph_values[j,i] <- log(ObjFun(c(x_seq[i], x_seq[j]))+rho_graph*p(c(x_seq[i], x_seq[j]))) #log scale
        #graph_values[j,i] <- ObjFun(c(x_seq[i], x_seq[j]))+rho_graph*p(c(x_seq[i], x_seq[j]))
    }
}

fig <- plot_ly(type = 'surface',
               x =seq(from=-4,to=4,length=n_grid),
               y =seq(from=-4,to=4,length=n_grid),
               z = graph_values)
#fig %>% add_markers(pm$x_opt[1],pm$x_opt[2],pm$f_opt[1])
fig
fig %>% add_markers(pm$x_opt[1],pm$x_opt[2],log(pm$f_opt[1])) #log scale
```

In [11]:

```
# Wykresy warstwic
contour(x_seq, x_seq, f_val)
contour(x_seq, x_seq, g_val, levels = seq(0, -5, by = -0.1), col = 'forestgreen', add =
T, drawlabels = F)
contour(x_seq, x_seq, g_val, levels = c(0), col = 'forestgreen', add = T, lwd = 2)
contour(x_seq, x_seq, h_val, levels = c(0), col = 'red', add = T, lwd = 2)
legend("bottomleft", legend=c("f(x)", "g(x)","h(x)"),col=c("black","green", "red"), lty
=1:1:1, cex=1.4)
```