Nonlinear Optimization Ex.2

Ariel Vishne 204149371

02 05 2022

Ex. 1 - Gradien Descent Method

We require:

$$f(x,y) = 100(y-x^2)^2 + (1-x)^2 = 100(y^2 - 2x^2y + x^4) + 1 - 2x + x^2 = 100x^4 + x^2 - 2x - 200x^2y + 100y^2 + 1$$

Defined as a function for later use in code we have:

```
f <- function(x,y){
 return( 100 * ((y - x ^ 2) ^ 2) + ((1 - x) ^ 2))
}
```

The gradient is:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 400x^3 + 2x - 2 - 400xy \\ -200x^2 + 200y \end{bmatrix}$$

Therefore the formula for gradient descent, for given $Z^n = (x,y)$ will be defined as:

$$Z^{n+1} = Z^n - \alpha \cdot \nabla f(Z^n) = \begin{bmatrix} x \\ y \end{bmatrix} - \alpha \cdot \begin{bmatrix} 400x^3 + 2x - 2 - 400xy \\ -200x^2 + 200y \end{bmatrix} = \begin{bmatrix} x - \alpha \cdot (400x^3 + 2x - 2 - 400xy) \\ y - \alpha \cdot (-200x^2 + 200y) \end{bmatrix}$$
$$= \begin{bmatrix} x - 2\alpha \cdot (200x^3 + x - 1 - 200xy) \\ y - 200\alpha \cdot (-x^2 + y) \end{bmatrix}$$

We will also create the gradient descent function to be used later in the code. This function recieves given $Z^n = (x,y)$ and α and computes the next iteration.

```
compute.gradient.f <- function(x,y){
  return(list
         (
           (2 * ((200 * (x^3)) +  x - 1 - (200 * y * x))),
           200 * ((-1 * (x ^ 2) + y))
              )
         )
}

next.iteration <- function(x,y,alpha){
  gradient <- compute.gradient.f(x,y)
  nabla.f.1 <- gradient[[1]]
  nabla.f.2 <- gradient[[2]]
  return(list
            (
              x - (alpha * nabla.f.1),
              y - (alpha * nabla.f.2)
              )
         )
}
```

We need to find the optimal α for this procedure. We will define a new function:

$$g(\alpha) = f(Z^n - \alpha \cdot \nabla f(Z^n))$$

We will use golden search.

```
g <- function(x,y,alpha){
  res <- next.iteration(x,y,alpha)
  x <- res[[1]]
  y <- res[[2]]
  return(f(x,y))
}

golden.ratio <- ((1 + sqrt(5)) / 2)

compute.alpha <- function(a, b,x,y, tol){
    d <- b - ((b-a) / golden.ratio)
    c <- a + ((b-a) / golden.ratio)

    while(abs(d - c) > tol){
      if (g(x,y,d) < g(x,y,c)){
        b <- c
      }
      else{
        a <- d
      }
      d <- b - ((b-a) / golden.ratio)
      c <- a + ((b-a) / golden.ratio)
    }
    return((a+b) / 2)
}
```

Implementation of gradient descent method is therefore as follows. Notice that at each iteration we use the optimal alpha through the golden search.

```
sdescent <- function(a0, tol, maxiter){

  x0 <- a0[1]
  x <- x0
  y0 <- a0[2]
  y <- y0
  a.t <- a0
  a.values <- matrix(0,ncol = 2, nrow = maxiter, byrow = TRUE)

  num.iter <- 1

  while(num.iter < maxiter){

    #print("iteration number")
    #print(num.iter)
    #print("x, y values")
    #print(round(x,6))
    #print(round(y,6))
    #print("function value at points")
    #print(round(f(x,y), 6))
    #print("================")

    a.values[num.iter,] <- c(x,y)

    num.iter <- num.iter + 1

    #print("Computing alpha")
    grad <- compute.gradient.f(x,y)
    #alpha <- compute.alpha(0,1,x,y,tol)
    alpha <- 0.0001
    #print("alpha value is")
    #print(alpha)
    #print("================")
    a.t.minus.one <- a.t

    a.t <- next.iteration(x,y,alpha)


    x <- a.t[[1]]
    y <- a.t[[2]]


    if (norm(as.numeric(c(x,y)) - as.numeric(a.t.minus.one), type = "2") < tol){
      break
    }

  }
  return(list(
    a.values,
    num.iter
  ))

}
```

What is left is to produce the graph with the results. We show the results with different resolutions.

1b Plotting Results

First setting - Good initialization

Our inital values are $a^0 = (0.95, 0.95)$. maxiter is 1000.

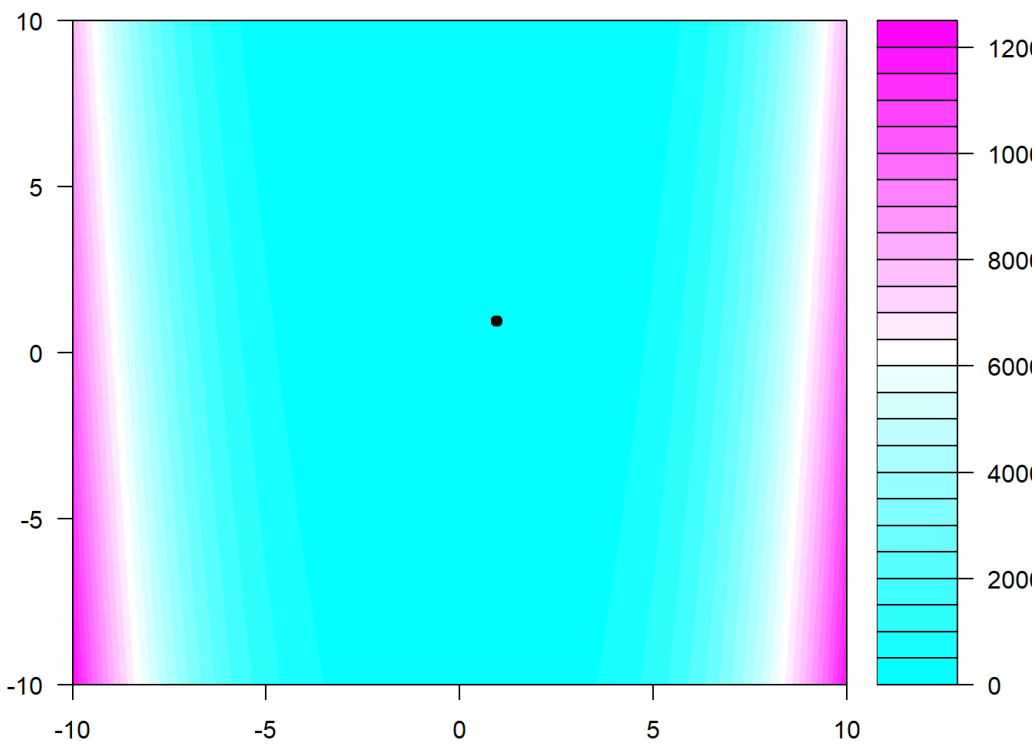
```
a0 <- c(0.95, 0.95)
tol <- 10^-5
maxiter <- 10000

res <- sdescent(a0, tol, maxiter)
values <- res[[1]]
values[maxiter,] <- NA

iters <- res[[2]]
print(paste("Iterations required:",iters))
```

```
## [1] "Iterations required: 57"
```

```
values <- values[1:iters-1,]
good.values <- values

# redefine f so to work with ContourFunctions library
f2 <- function(r){
 return( 100 * ((r[2] - r[1] ^ 2) ^ 2) + ((1 - r[1]) ^ 2))
}

print("ZOOM OUT values -10 to 10")
```

```
## [1] "ZOOM OUT values -10 to 10"
```

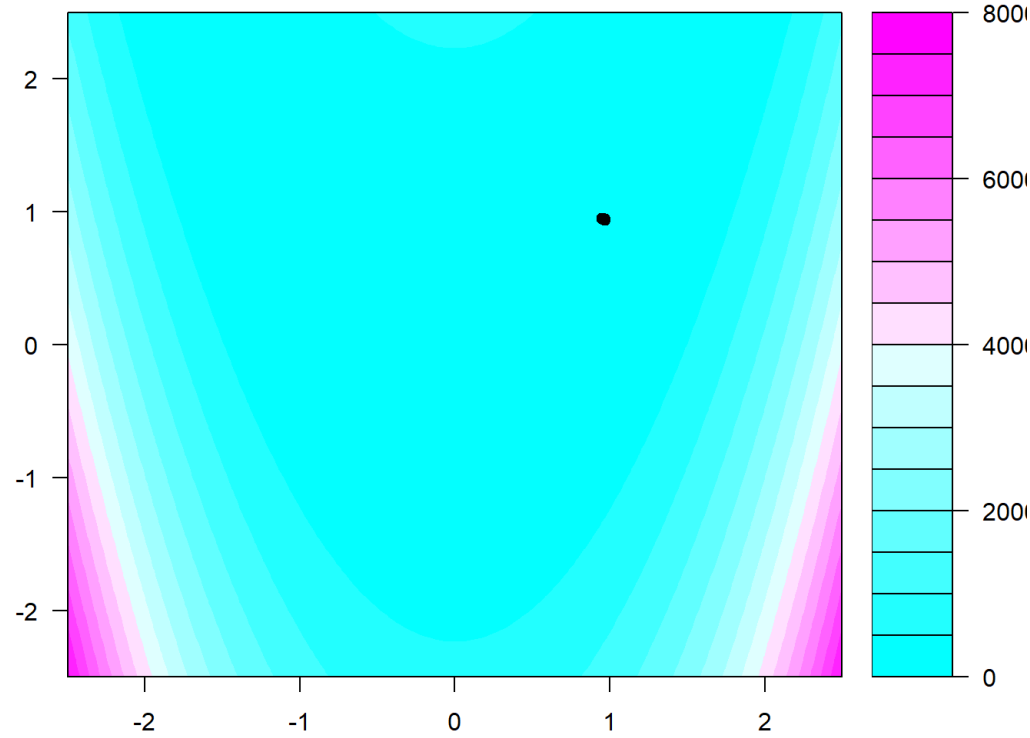
```
cf_func(f2, xlim = c(-10, 10), ylim = c(-10, 10), bar = TRUE, pts = values)
```




```
print("ZOOM OUT values -2.5 to 2.5")
```

```
## [1] "ZOOM OUT values -2.5 to 2.5"
```

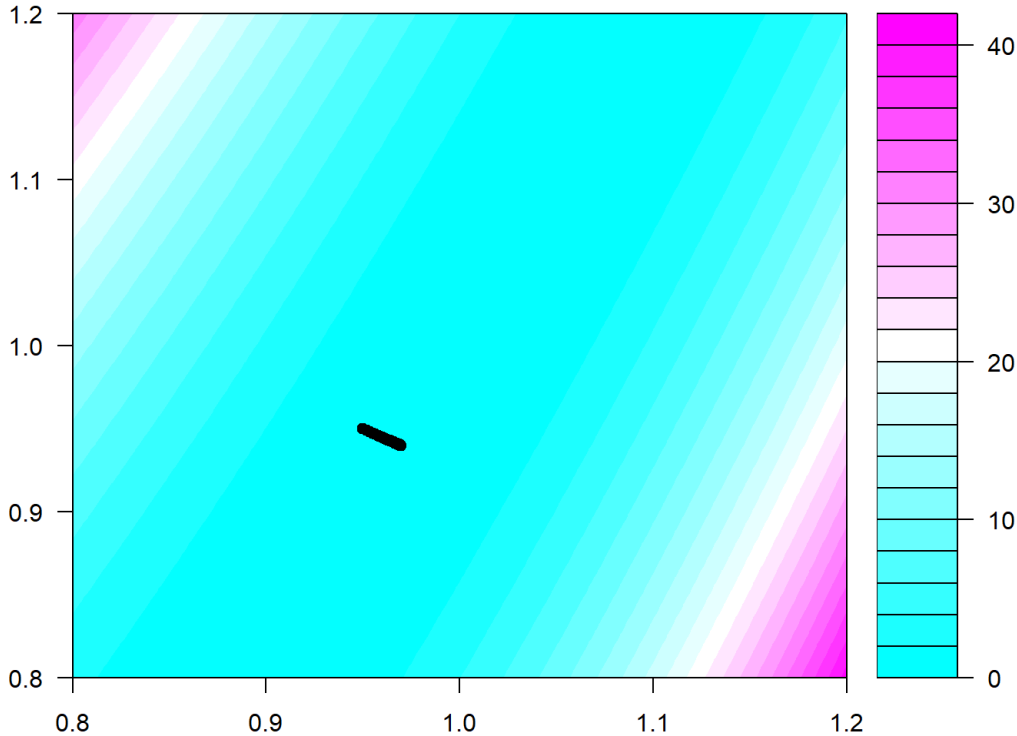
```
cf_func(f2, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5), bar = TRUE, pts = values)
```




```
print("ZOOM IN values 0.8-1.2")
```

```
## [1] "ZOOM IN values 0.8-1.2"
```

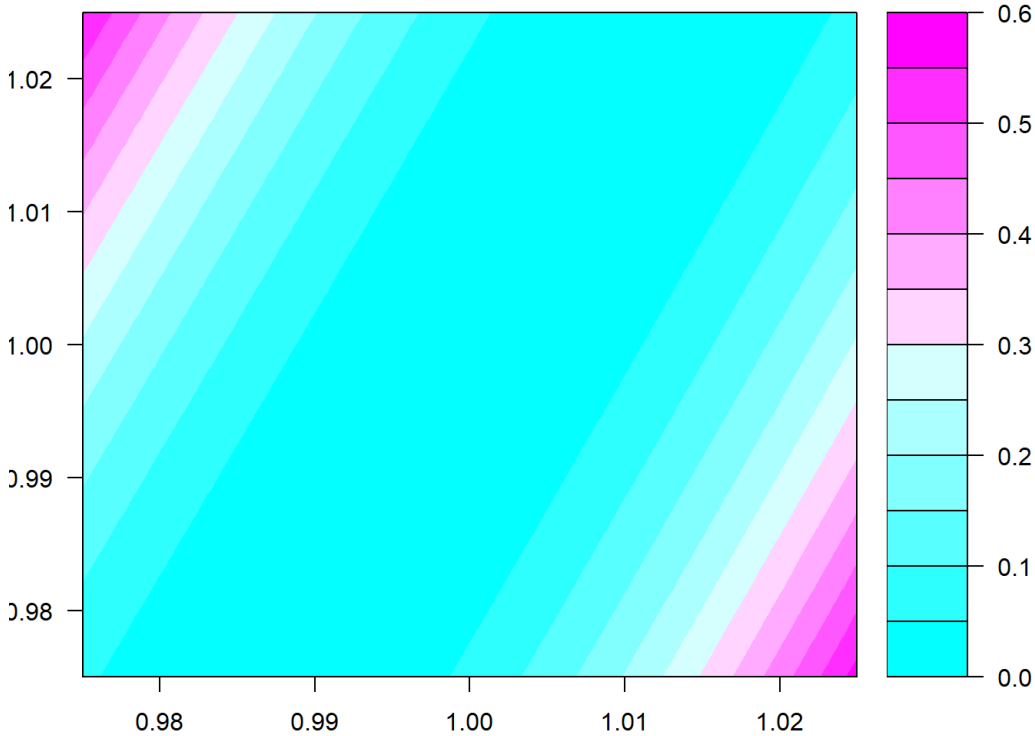
```
cf_func(f2, xlim = c(0.8, 1.2), ylim = c(0.8, 1.2), bar = TRUE, pts = values)
```




```
print("ZOOM More IN values 0.975-1.025")
```

```
## [1] "ZOOM More IN values 0.975-1.025"
```

```
cf_func(f2, xlim = c(0.975, 1.025), ylim = c(0.975, 1.025), bar = TRUE, pts = values)
```



Second setting - average initialization

Our inital values are $a^0 = (5, 5)$. maxiter is 10000.

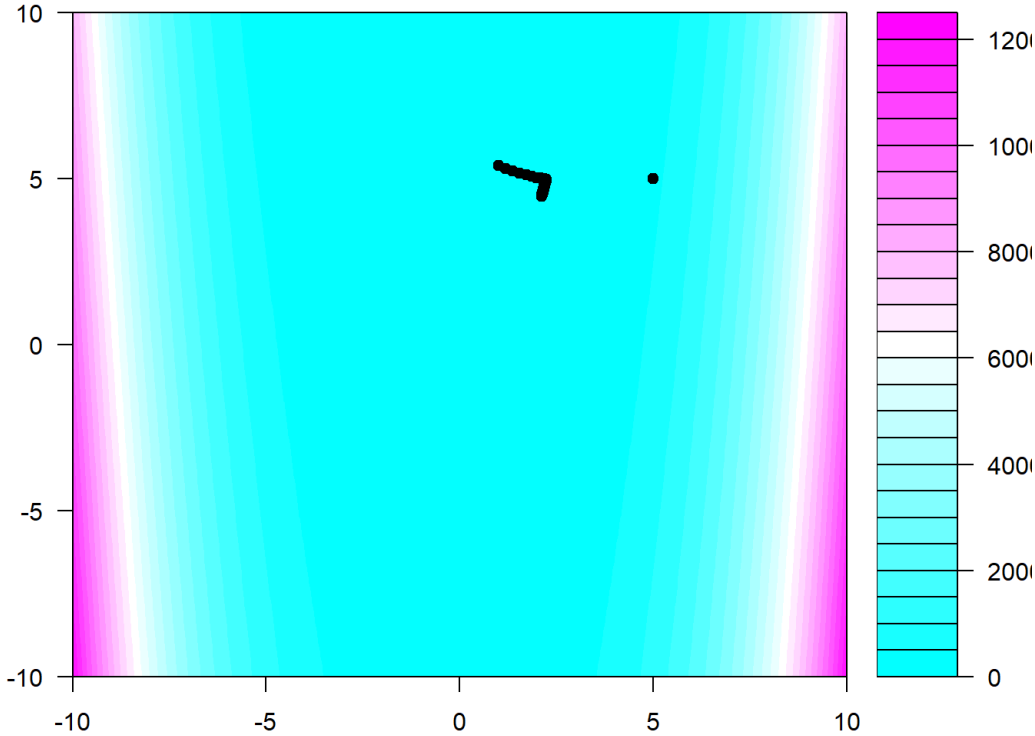
```
a0 <- c(5,5)
tol <- 10^-5
maxiter <- 10000

res <- sdescent(a0, tol, maxiter)
values <- res[[1]]
values[maxiter,] <- NA
iters <- res[[2]]
print(paste("Iterations required:",iters))
```

```
## [1] "Iterations required: 10000"
```

```
print("ZOOM OUT values -10 to 10")
```

```
## [1] "ZOOM OUT values -10 to 10"
```

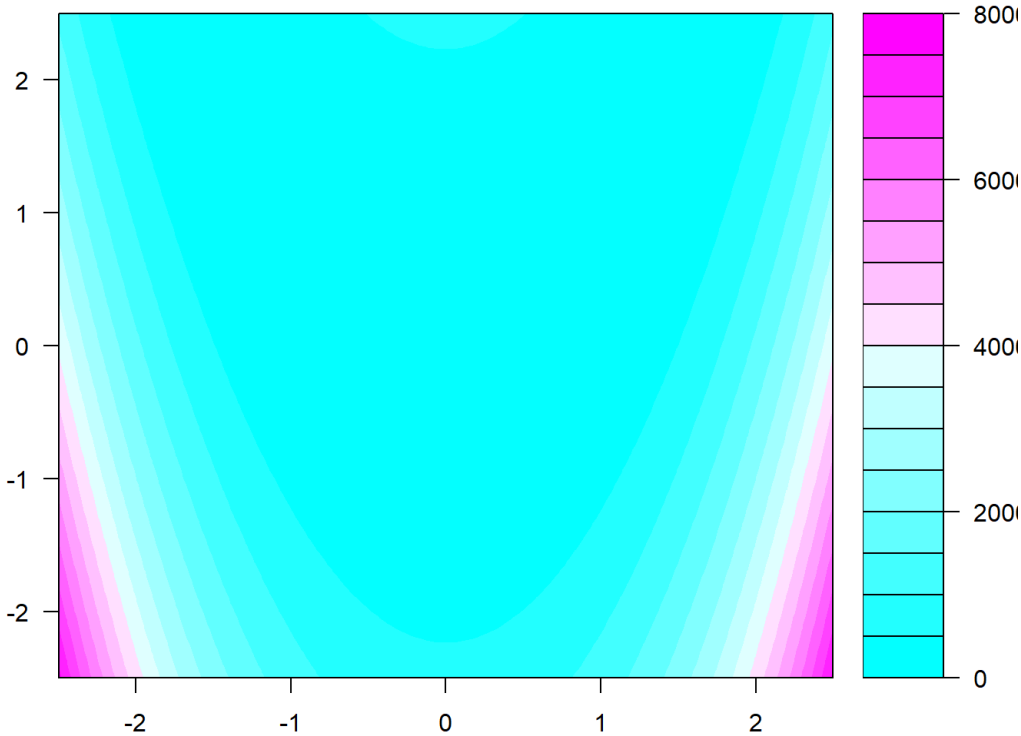
```
cf_func(f2, xlim = c(-10, 10), ylim = c(-10, 10), bar = TRUE, pts = values)
```




```
print("ZOOM OUT values -2.5 to 2.5")
```

```
## [1] "ZOOM OUT values -2.5 to 2.5"
```

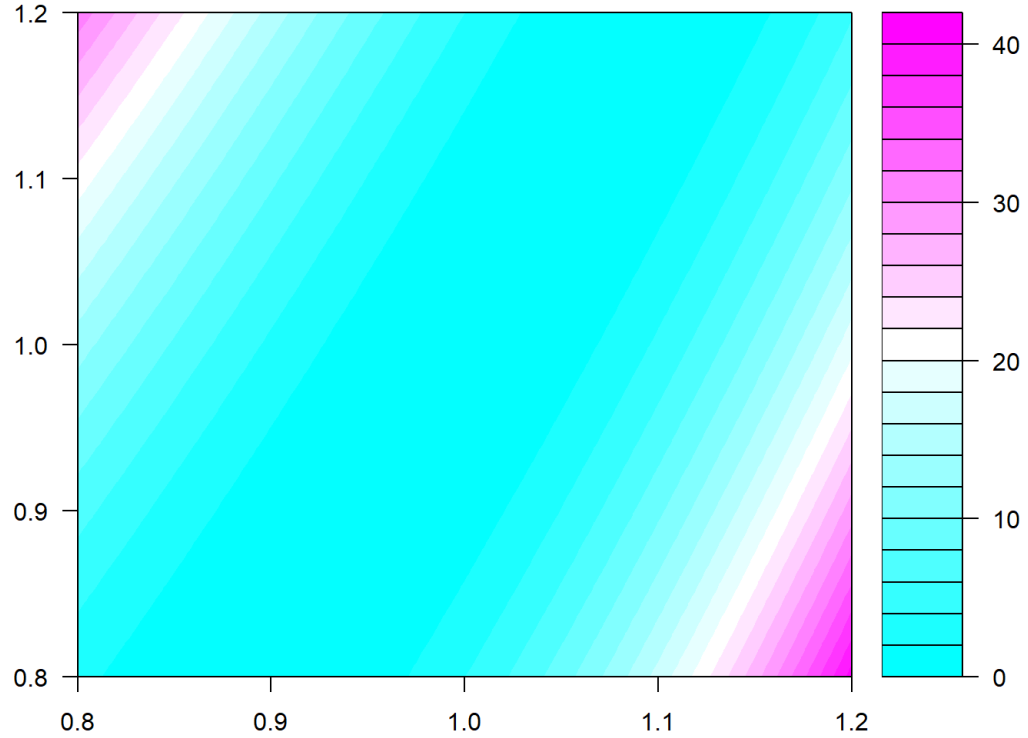
```
cf_func(f2, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5), bar = TRUE, pts = values)
```




```
print("ZOOM IN values 0.8-1.2")
```

```
## [1] "ZOOM IN values 0.8-1.2"
```

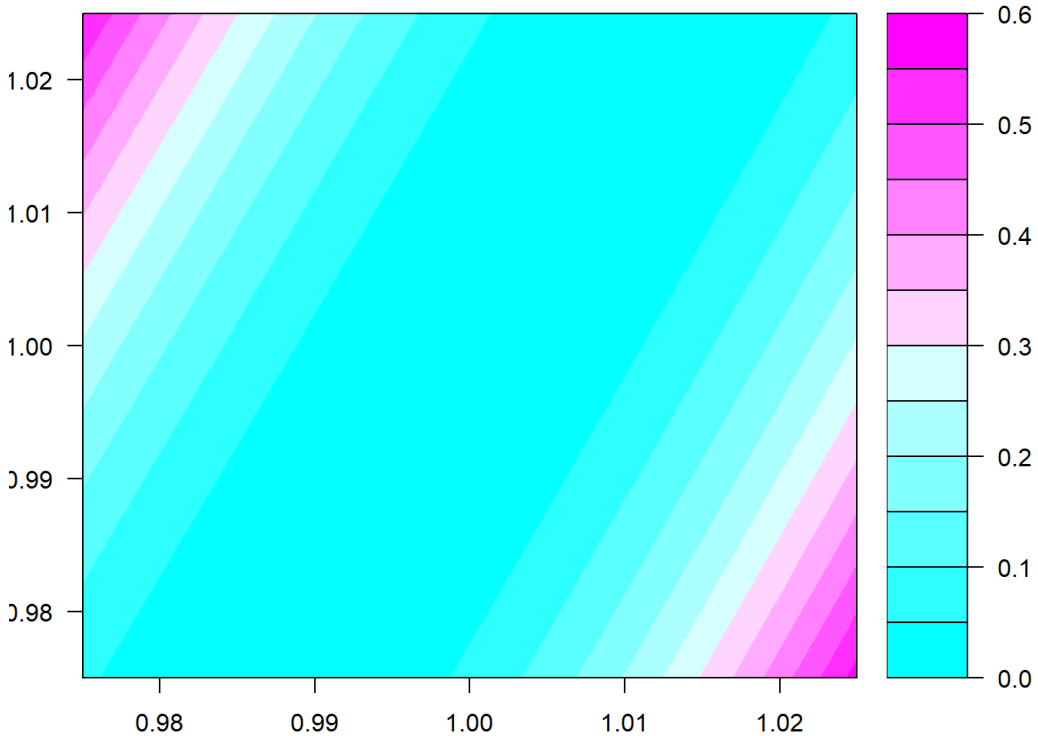
```
cf_func(f2, xlim = c(0.8, 1.2), ylim = c(0.8, 1.2), bar = TRUE, pts = values)
```




```
print("ZOOM More IN values 0.975-1.025")
```

```
## [1] "ZOOM More IN values 0.975-1.025"
```

```
cf_func(f2, xlim = c(0.975, 1.025), ylim = c(0.975, 1.025), bar = TRUE, pts = values)
```



Third setting - bad initialization

Our inital values are $a^0 = (7, 7)$. maxiter is 10000.

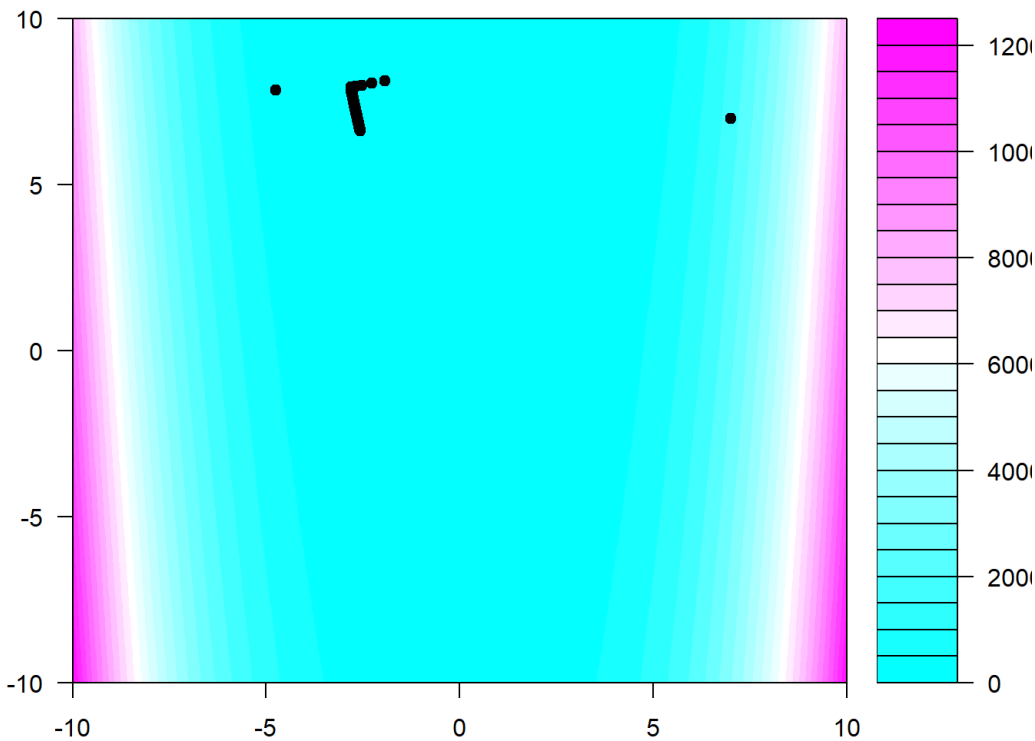
```
a0 <- c(7,7)
tol <- 10^-5
maxiter <- 10000

res <- sdescent(a0, tol, maxiter)
values <- res[[1]]
values[maxiter,] <- NA
iters <- res[[2]]
print(paste("Iterations required:",iters))
```

```
## [1] "Iterations required: 10000"
```

```
print("ZOOM OUT values -10 to 10")
```

```
## [1] "ZOOM OUT values -10 to 10"
```

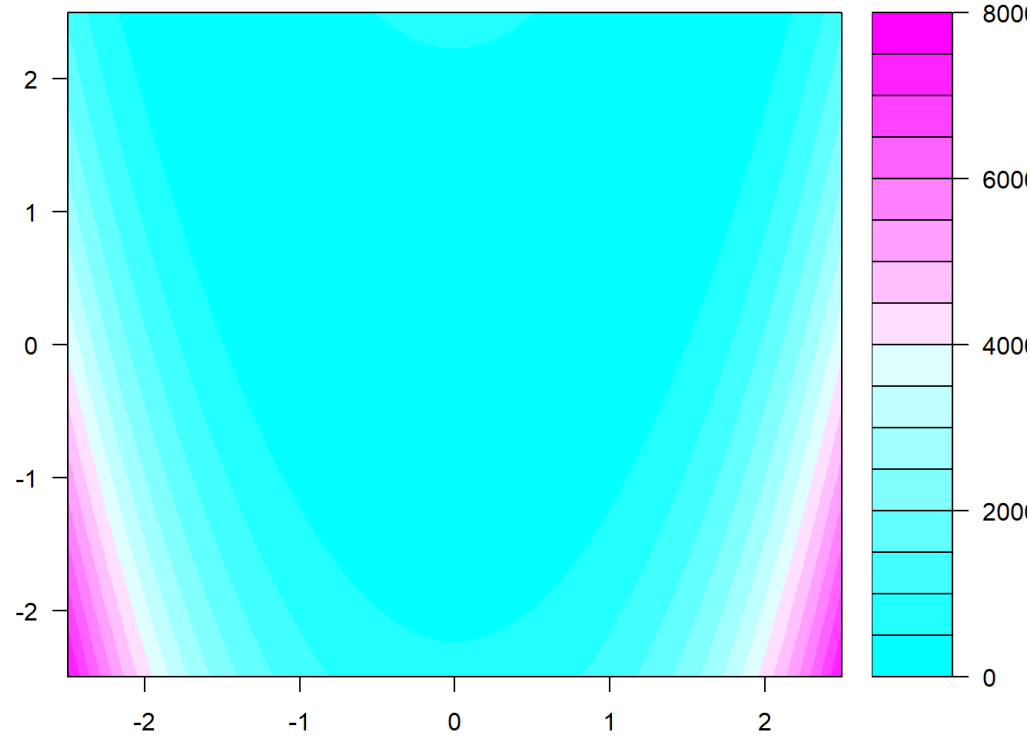
```
cf_func(f2, xlim = c(-10, 10), ylim = c(-10, 10), bar = TRUE, pts = values)
```




```
print("ZOOM OUT values -2.5 to 2.5")
```

```
## [1] "ZOOM OUT values -2.5 to 2.5"
```

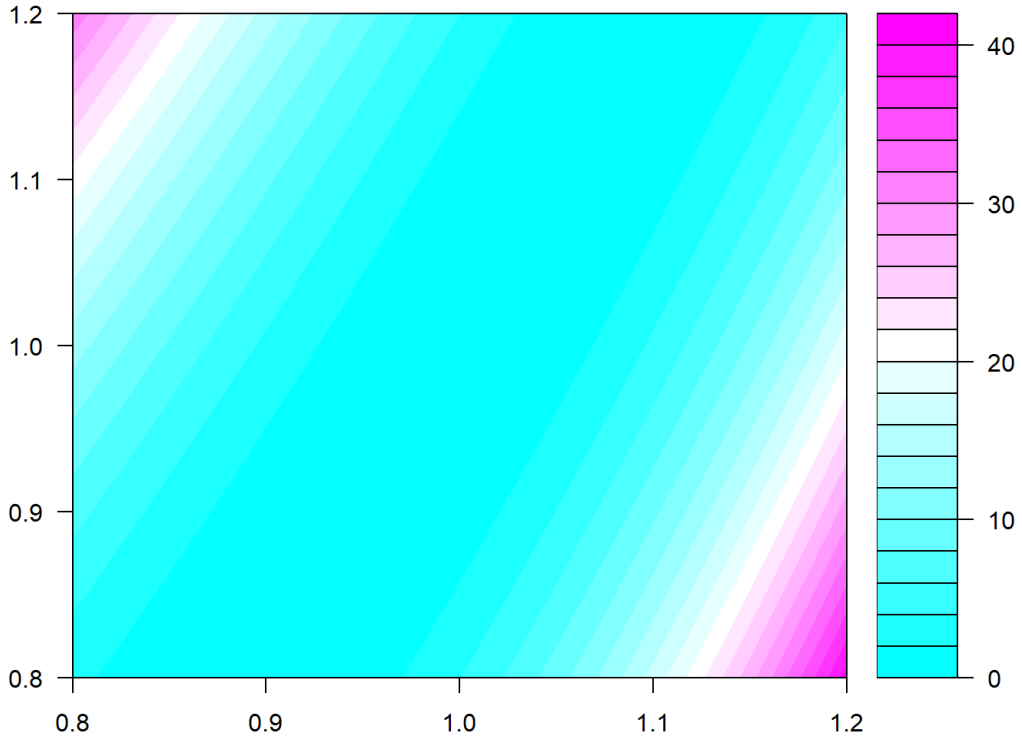
```
cf_func(f2, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5), bar = TRUE, pts = values)
```




```
print("ZOOM IN values 0.8-1.2")
```

```
## [1] "ZOOM IN values 0.8-1.2"
```

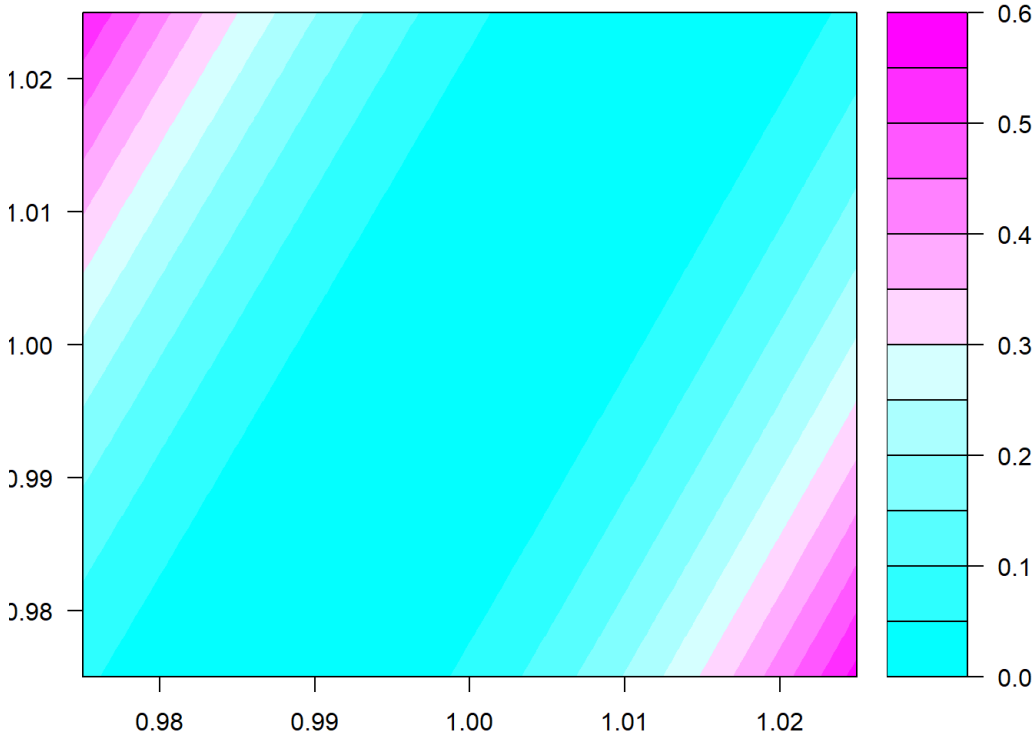
```
cf_func(f2, xlim = c(0.8, 1.2), ylim = c(0.8, 1.2), bar = TRUE, pts = values)
```




```
print("ZOOM More IN values 0.975-1.025")
```

```
## [1] "ZOOM More IN values 0.975-1.025"
```

```
cf_func(f2, xlim = c(0.975, 1.025), ylim = c(0.975, 1.025), bar = TRUE, pts = values)
```



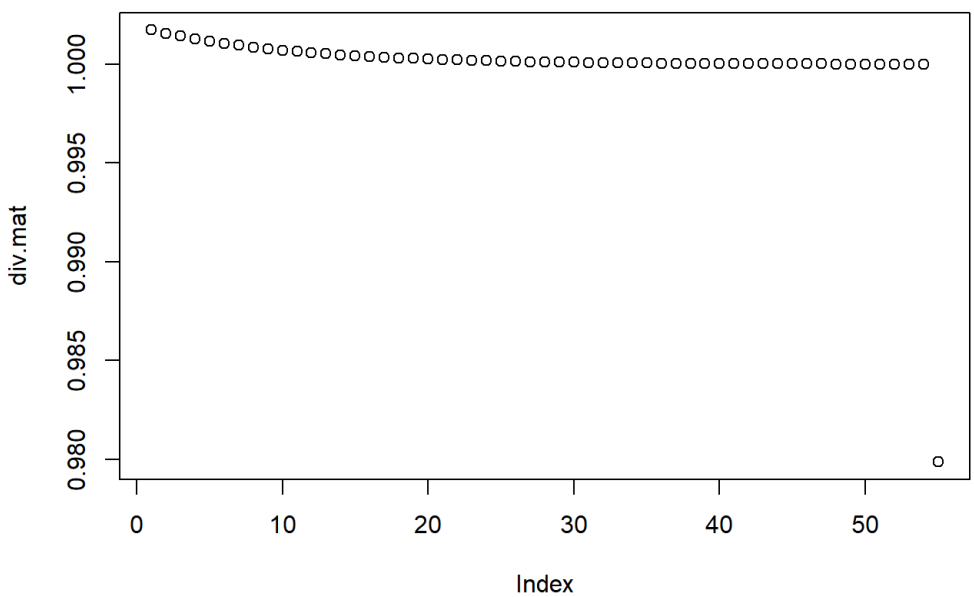
1c Speed of Convergence

We can see that the convergence is very slow, it is linear in relation with number of iterations. We can see that when close to the point of convergence there is no more 'jumps' in the function, and the convergence is very gradual and takes about 56 iterations at the starting point $(0, 0$. For the average initialization it took about 9500 iterations to converge. On the bad setting even that was not enough. We will take the first (good) initialization as further investigation. We will compute this numerically with the following equation:

$$C = \lim_{n->\infty} \frac{x_{n+1} - \bar{x}}{x_n - \bar{x}} = \lim_{n->\infty} \frac{||x_{n+1} - \bar{x}||^2}{||x_n - \bar{x}||^2}$$

where $\bar{x}$ is the (now) known minimum point $(1, 1)$.

```
xbar <- as.numeric(c(1,1))
norms.mat <- rep(0, dim(good.values)[1])
for (i in 1:dim(good.values)[1]){
norms.mat[i] <- norm(as.numeric(good.values[i,]) - xbar, type = "2")
}
div.mat <- rep(0, dim(good.values)[1]-1)
for (i in 1:dim(good.values)[1]-1){
  div.mat[i] <- good.values[i+2] / good.values[i+1]
}

plot(div.mat)
```



As we can see the limit is 1, i.e. the convergence rate is linear.

Ex. 2 - Newton Method

For the Newton method we require computing the Hessian $H(f)^-1$ as follows:

$$H(f) = \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial^2 x} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial x \partial y} & \frac{\partial^2 f(x,y)}{\partial^2 y} \end{bmatrix} = \begin{bmatrix} 1200x^2 + 2 - 400y & -400x \\ -400x & 200 \end{bmatrix}$$

The determinant of the Hessian is given by:

$$\det(H(f)) = \begin{vmatrix} 1200x^2 + 2 - 400y & -400x \\ -400x & 200 \end{vmatrix} = 200 * (1200x^2 + 2 - 400y) - (-400x * -400x) = 400(600x^2 + 1 - 200y) - 400^2x^2 =$$
$$400[600x^2 + 1 - 200y - 400x^2] = 400[200x^2 - 200y + 1]$$

And therefore the inverse is given by:

$$H(f)^{-1} = \frac{1}{400(200x^2 - 200y + 1)} \begin{bmatrix} 200 & 400x \\ 400x & 1200x^2 + 2 - 400y \end{bmatrix}$$

The whole process of the Newton Method will therefore be as follow, for some point $Z^n = (x, y)$:

$$\begin{aligned} Z^{n+1} &= Z^n - H(f)^{-1}(Z^n) \bullet \nabla f(Z^n) \\ &= \begin{bmatrix} x \\ y \end{bmatrix} - \frac{1}{400(200x^2 - 200y + 1)} \begin{bmatrix} 200 & 400x \\ 400x & 1200x^2 + 2 - 400y \end{bmatrix} \begin{bmatrix} 400x^3 + 2x - 2 - 400xy \\ -200x^2 + 200y \end{bmatrix} \\ &= \begin{bmatrix} x \\ y \end{bmatrix} - \frac{1}{100(200x^2 - 200y + 1)} \begin{bmatrix} 100 & 200x \\ 200x & 600x^2 + 1 - 200y \end{bmatrix} \begin{bmatrix} 200x^3 + x - 1 - 200xy \\ -100x^2 + 100y \end{bmatrix} \\ &= \begin{bmatrix} x \\ y \end{bmatrix} - \frac{1}{100(200x^2 - 200y + 1)} \begin{bmatrix} 100(200x^3 + x - 1 - 200xy) + 200x(-100x^2 + 100y) \\ 200x(200x^3 + x - 1 - 200xy) + (600x^2 + 1 - 200y)(-100x^2 + 100y) \end{bmatrix} \\ &= \begin{bmatrix} x \\ y \end{bmatrix} - \frac{1}{(200x^2 - 200y + 1)} \begin{bmatrix} 200x^3 + x - 1 - 200xy - 200x^3 + 200xy \\ 400x^4 + 2x^2 - 2x - 400x^2y - 600x^4 - x^2 + 200x^2y + 600x^2y + y - 200y^2 \end{bmatrix} \\ &= \begin{bmatrix} x \\ y \end{bmatrix} - \frac{1}{(200x^2 - 200y + 1)} \begin{bmatrix} x - 1 \\ -200x^4 + x^2 - 2x + 400x^2y + y - 200y^2 \end{bmatrix} \end{aligned}$$

So we can implement the following code:

```
newton.iteration <- function(x,y){
  mat <- as.numeric(list(x,y)) - (1 / (200 * x^2 - 200 * y + 1)) * as.numeric(list(x-1, -200 * x^4 + x^2 - 2 * x + 400 * x^2
* y + y - 200 * y^2))
  return(mat)
}
```

Now we can implement the whole Newton method as follows:

```
newton <- function(a0, tol, maxiter){

  x0 <- a0[1]
  x <- x0
  y0 <- a0[2]
  y <- y0
  a.t <- a0
  a.values <- matrix(0,ncol = 2, nrow = maxiter, byrow = TRUE)

  num.iter <- 0

  while(num.iter < maxiter){
    #print("iteration number")
    #print(num.iter)
    #print("x, y values")
    #print(round(x,6))
    #print(round(y,6))
    #print("function value at points")
    #print(round(f(x,y), 6))
    #print("================")

    num.iter <- num.iter + 1
    a.values[num.iter,] <- c(x,y)
    a.t.minus.one <- a.t

    a.t <- newton.iteration(x,y)

    x <- a.t[[1]]
    y <- a.t[[2]]


    if (norm(as.numeric(c(x,y)) - as.numeric(a.t.minus.one), type = "2") < tol){
      break
    }

  }
  return(list(
    a.values,
    num.iter
  ))

}
```

2b plotting examples

First

First setting - Good initialization

Our inital values are $a^0 = (0.95, 0.95)$. maxiter is 1000.

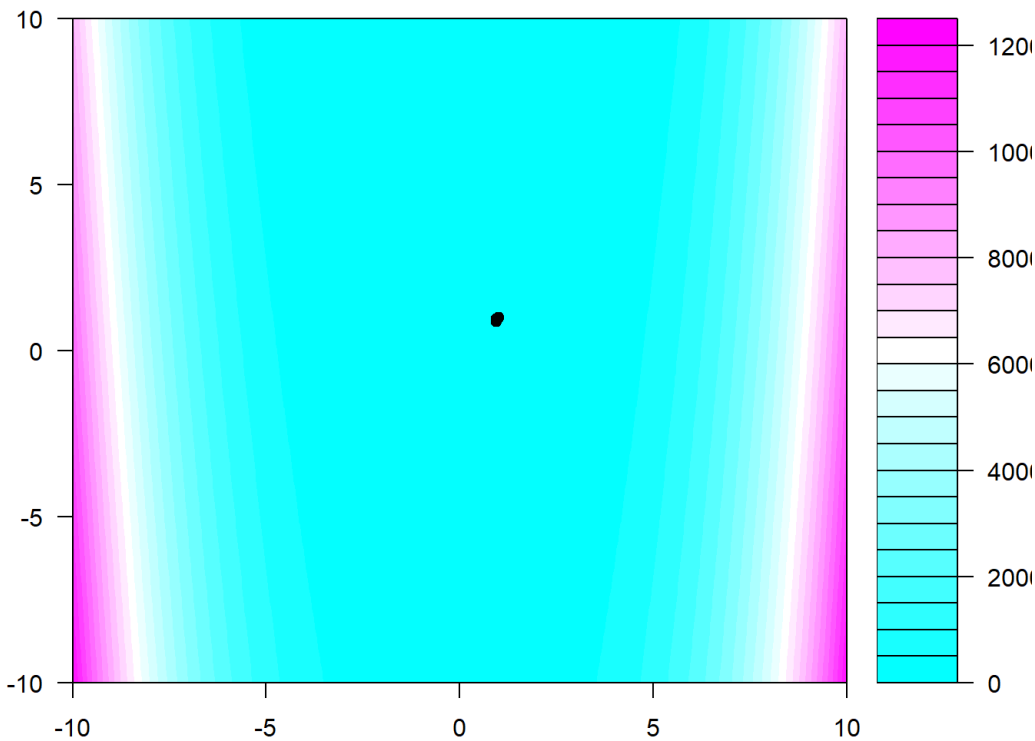
```
a0 <- c(0.95, 0.95)
tol <- 10^-5
maxiter <- 1000

res <- newton(a0, tol, maxiter)
values <- res[[1]]
values[maxiter,] <- NA
iters <- res[[2]]
values <- values[1:iters-1,]
good.values <- values
print(paste("Iterations required:",iters))
```

```
## [1] "Iterations required: 5"
```

```
print("ZOOM OUT values -10 to 10")
```

```
## [1] "ZOOM OUT values -10 to 10"
```

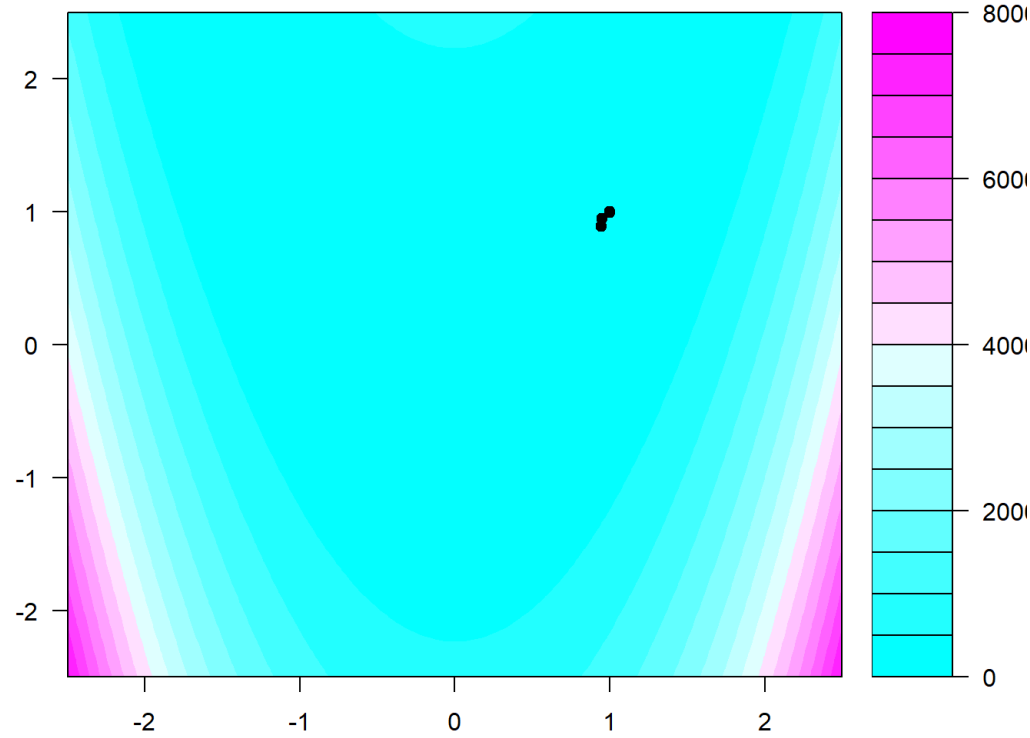
```
cf_func(f2, xlim = c(-10, 10), ylim = c(-10, 10), bar = TRUE, pts = values)
```




```
print("ZOOM OUT values -2.5 to 2.5")
```

```
## [1] "ZOOM OUT values -2.5 to 2.5"
```

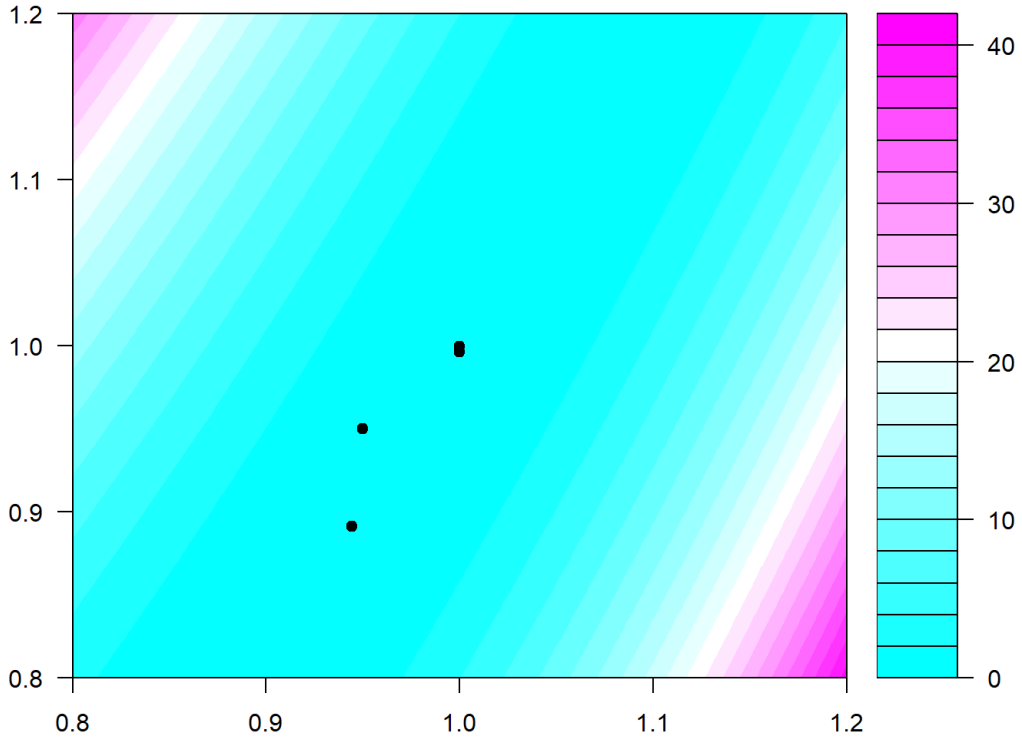
```
cf_func(f2, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5), bar = TRUE, pts = values)
```




```
print("ZOOM IN values 0.8-1.2")
```

```
## [1] "ZOOM IN values 0.8-1.2"
```

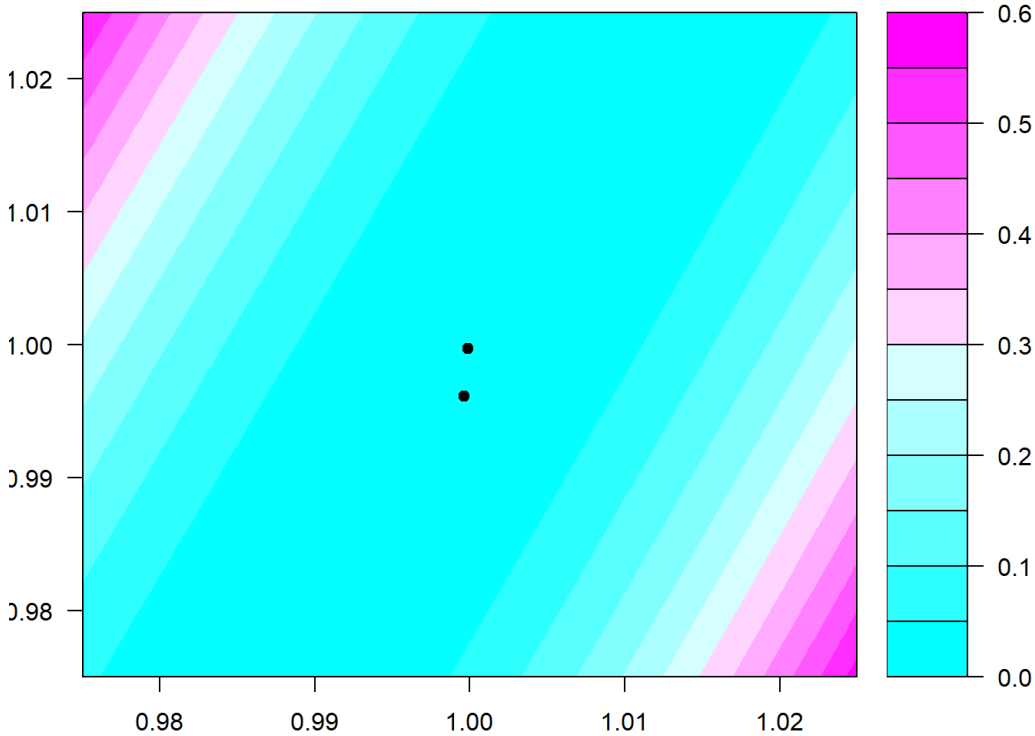
```
cf_func(f2, xlim = c(0.8, 1.2), ylim = c(0.8, 1.2), bar = TRUE, pts = values)
```




```
print("ZOOM More IN values 0.975-1.025")
```

```
## [1] "ZOOM More IN values 0.975-1.025"
```

```
cf_func(f2, xlim = c(0.975, 1.025), ylim = c(0.975, 1.025), bar = TRUE, pts = values)
```



Second setting - average

initialization Our inital values are $a^0 = (5, 5)$. maxiter is 10000.

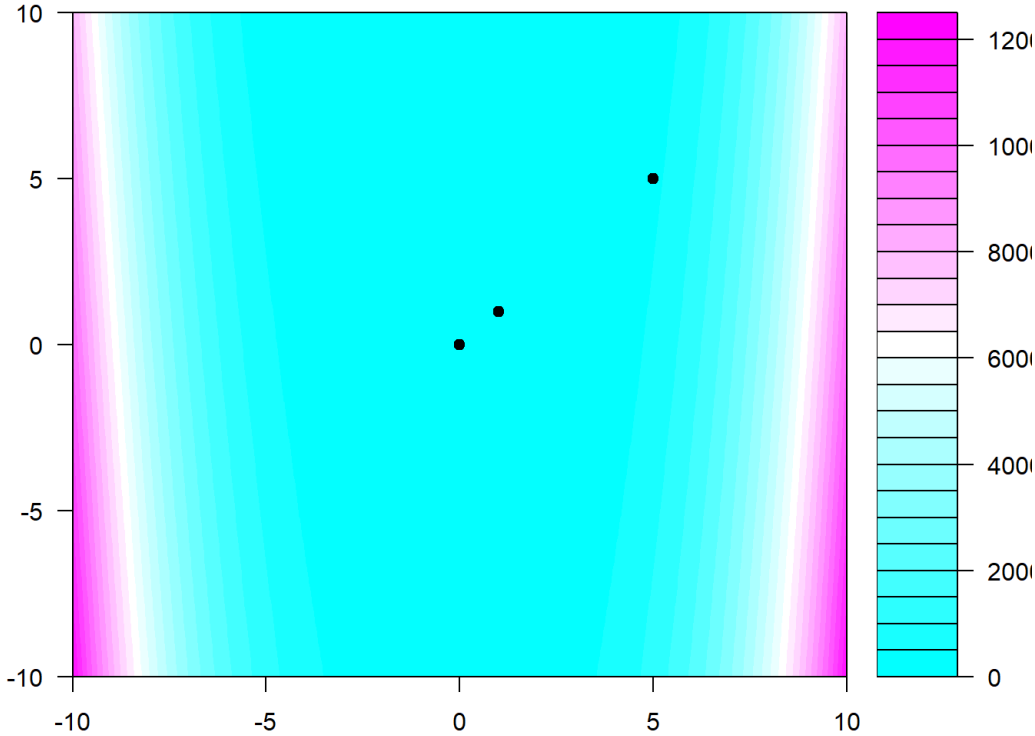
```
a0 <- c(5,5)
tol <- 10^-5
maxiter <- 1000

res <- newton(a0, tol, maxiter)
values <- res[[1]]
values[maxiter,] <- NA
iters <- res[[2]]
print(paste("Iterations required:",iters))
```

```
## [1] "Iterations required: 5"
```

```
print("ZOOM OUT values -10 to 10")
```

```
## [1] "ZOOM OUT values -10 to 10"
```

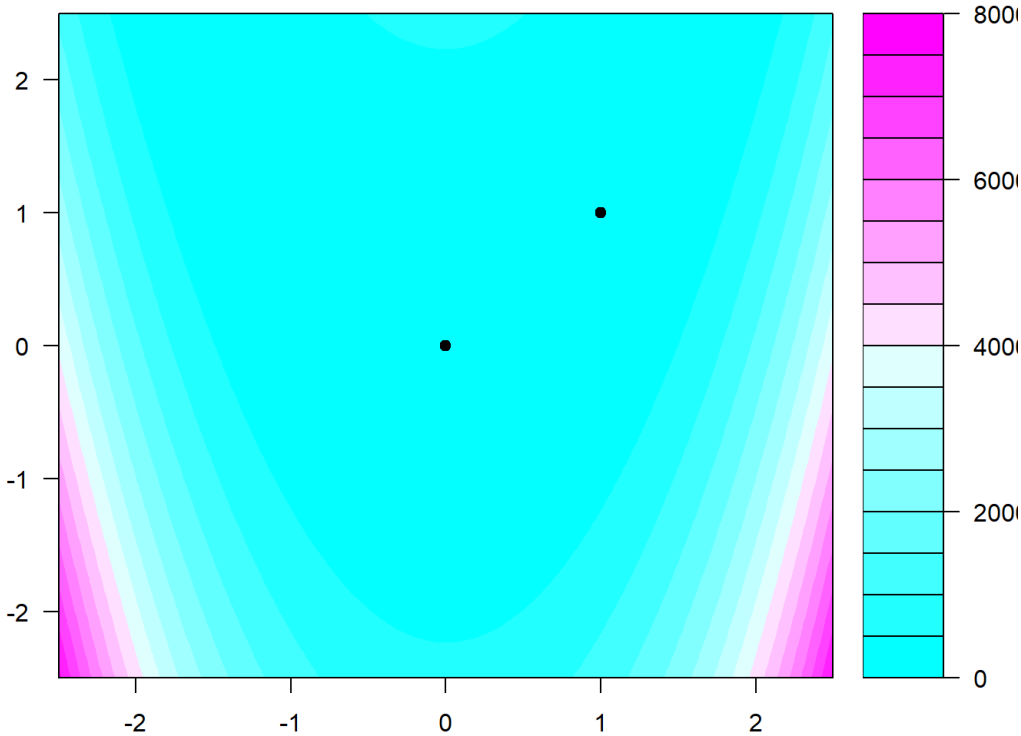
```
cf_func(f2, xlim = c(-10, 10), ylim = c(-10, 10), bar = TRUE, pts = values)
```




```
print("ZOOM OUT values -2.5 to 2.5")
```

```
## [1] "ZOOM OUT values -2.5 to 2.5"
```

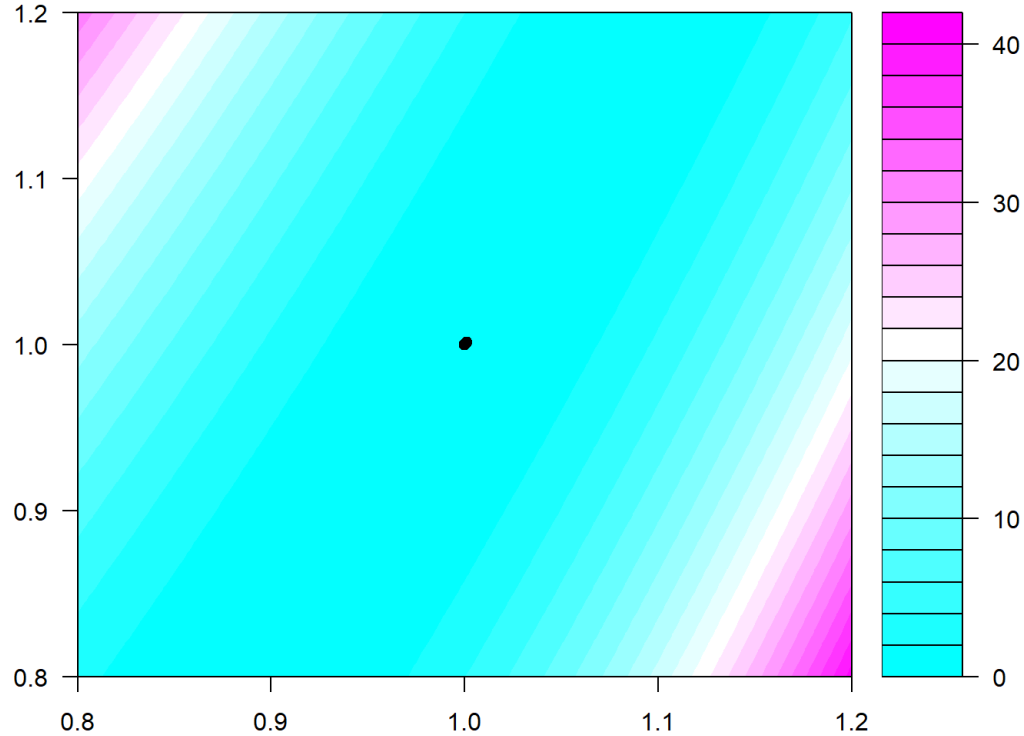
```
cf_func(f2, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5), bar = TRUE, pts = values)
```




```
print("ZOOM IN values 0.8-1.2")
```

```
## [1] "ZOOM IN values 0.8-1.2"
```

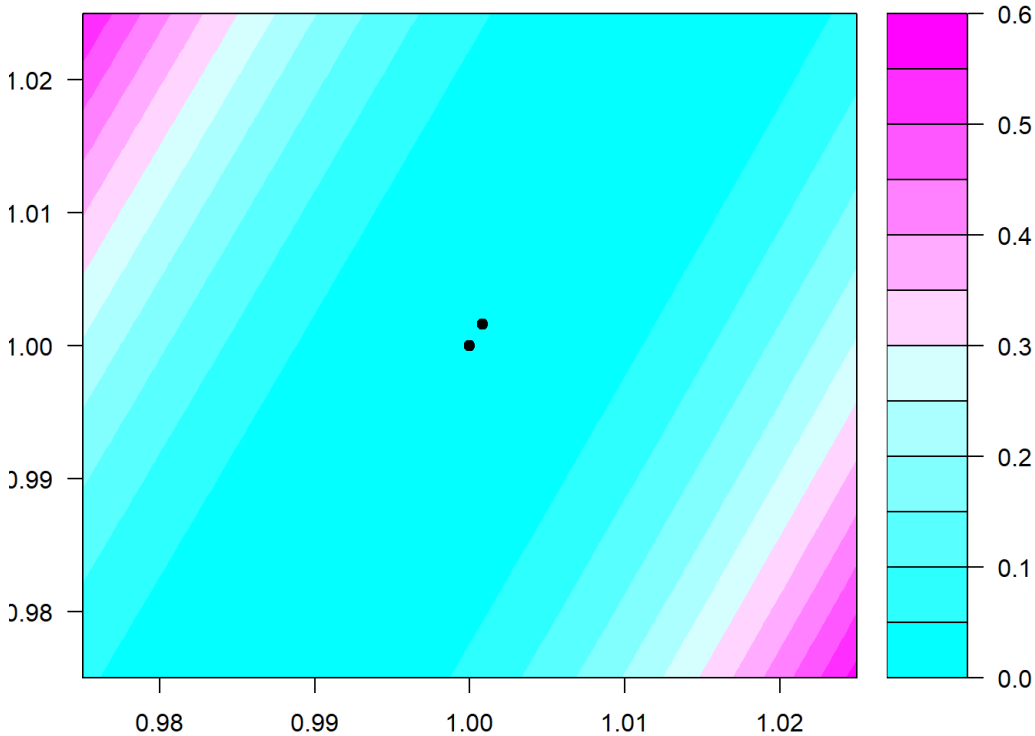
```
cf_func(f2, xlim = c(0.8, 1.2), ylim = c(0.8, 1.2), bar = TRUE, pts = values)
```




```
print("ZOOM More IN values 0.975-1.025")
```

```
## [1] "ZOOM More IN values 0.975-1.025"
```

```
cf_func(f2, xlim = c(0.975, 1.025), ylim = c(0.975, 1.025), bar = TRUE, pts = values)
```



Third setting - bad initialization

Our inital values are $a^0 = (-10, 10)$. maxiter is 1000.

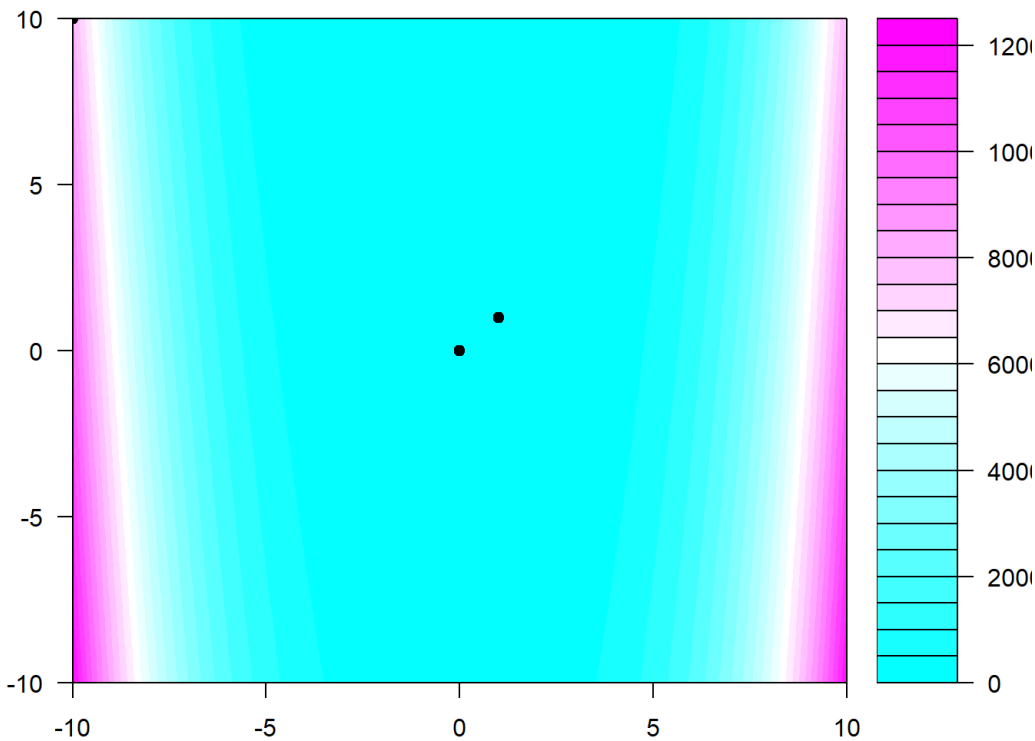
```
a0 <- c(-10,10)
tol <- 10^-5
maxiter <- 1000

res <- newton(a0, tol, maxiter)
values <- res[[1]]
values[maxiter,] <- NA
iters <- res[[2]]
print(paste("Iterations required:",iters))
```

```
## [1] "Iterations required: 5"
```

```
print("ZOOM OUT values -10 to 10")
```

```
## [1] "ZOOM OUT values -10 to 10"
```

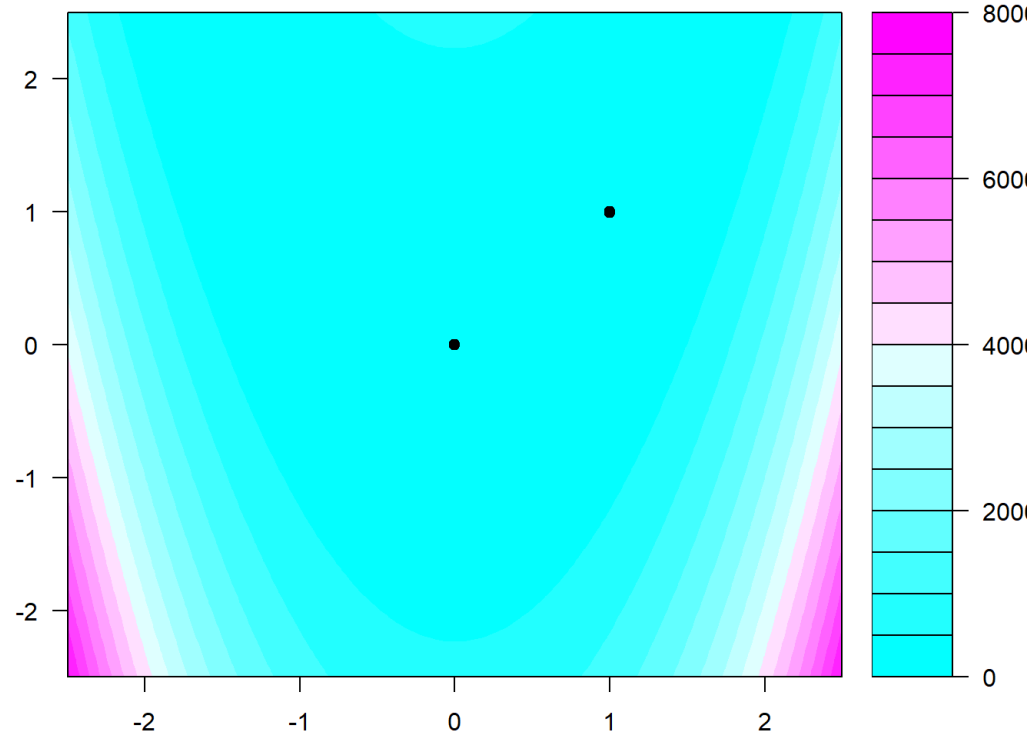
```
cf_func(f2, xlim = c(-10, 10), ylim = c(-10, 10), bar = TRUE, pts = values)
```




```
print("ZOOM OUT values -2.5 to 2.5")
```

```
## [1] "ZOOM OUT values -2.5 to 2.5"
```

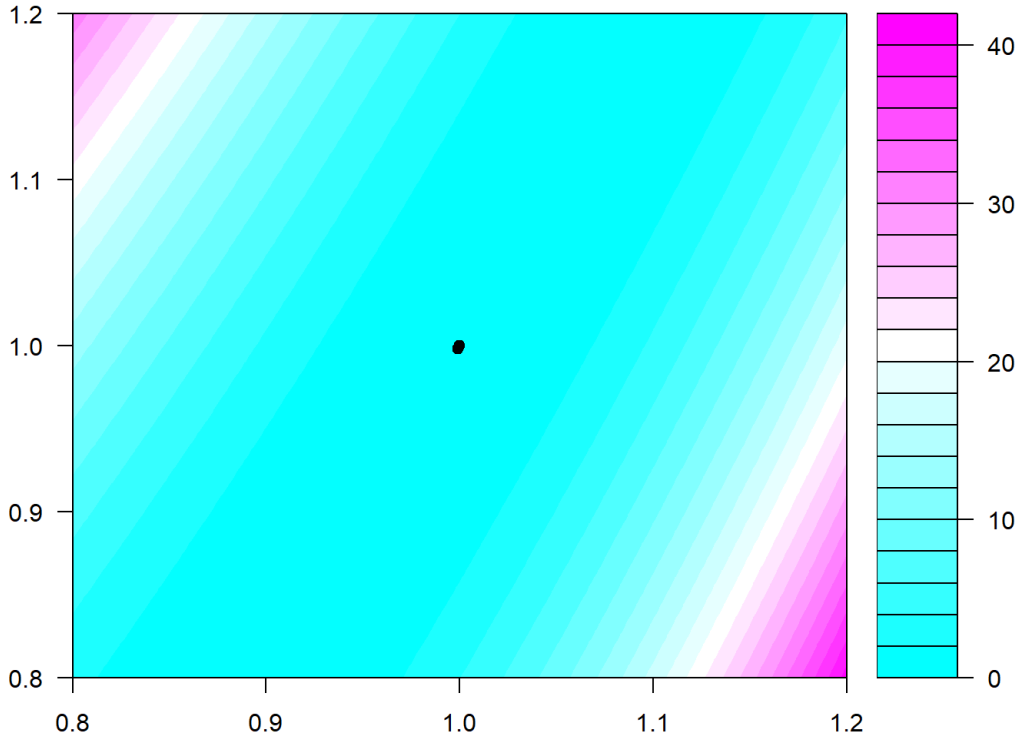
```
cf_func(f2, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5), bar = TRUE, pts = values)
```




```
print("ZOOM IN values 0.8-1.2")
```

```
## [1] "ZOOM IN values 0.8-1.2"
```

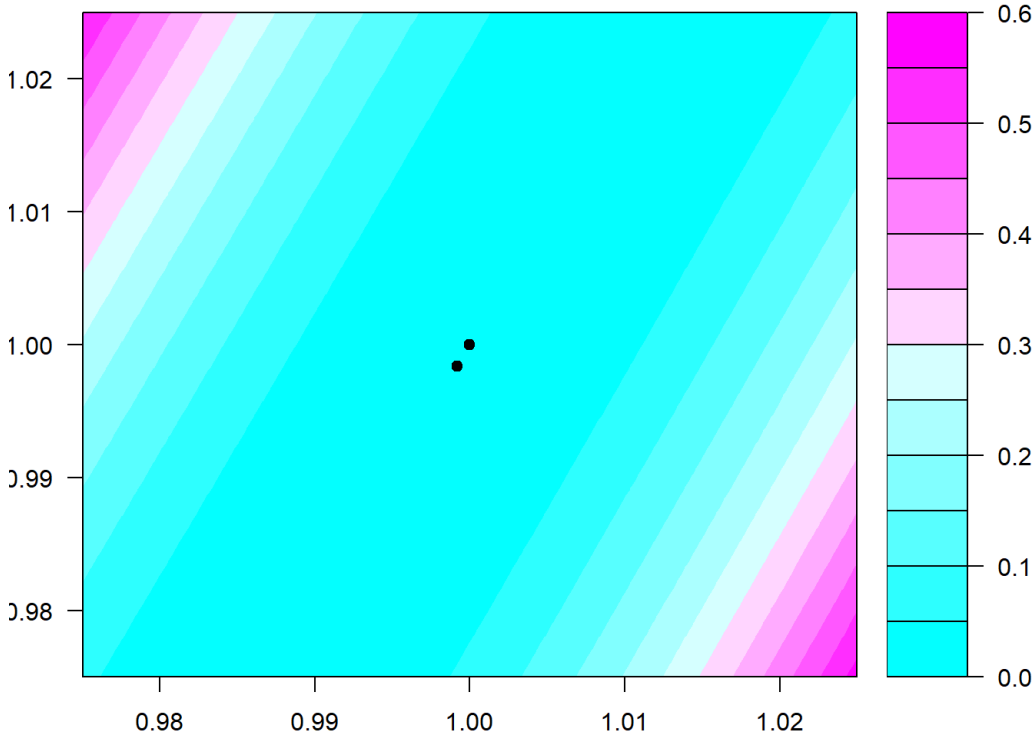
```
cf_func(f2, xlim = c(0.8, 1.2), ylim = c(0.8, 1.2), bar = TRUE, pts = values)
```




```
print("ZOOM More IN values 0.975-1.025")
```

```
## [1] "ZOOM More IN values 0.975-1.025"
```

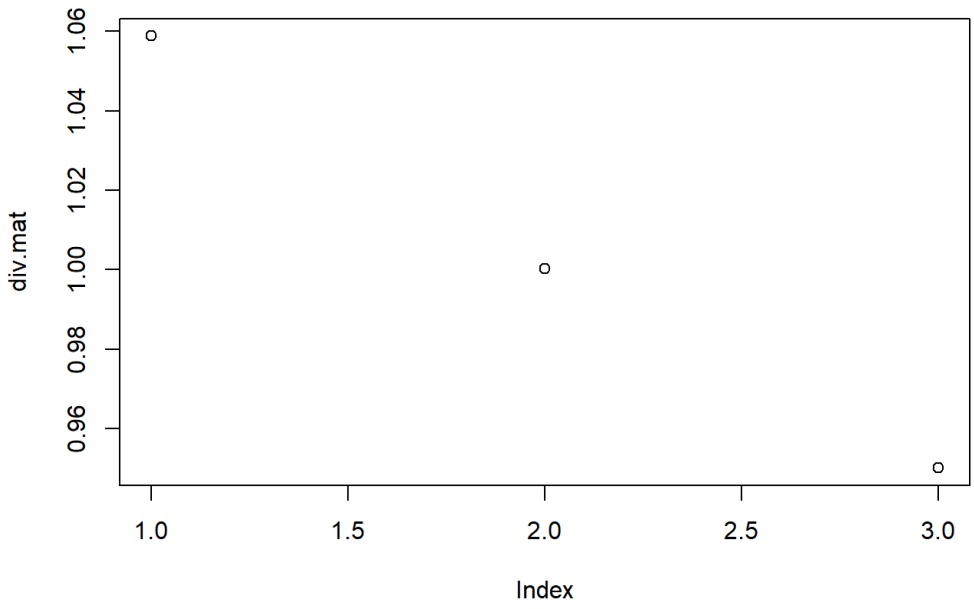
```
cf_func(f2, xlim = c(0.975, 1.025), ylim = c(0.975, 1.025), bar = TRUE, pts = values)
```



2c

We can see that the convergence is much faster with Newton process, even with a higher tolerance the convergence happens faster and no more than 6 iterations were required. Like in 1c we will use the good initialization values to look at the divergence rate.

```
xbar <- as.numeric(c(1,1))
norms.mat <- rep(0, dim(good.values)[1])
for (i in 1:dim(good.values)[1]){
norms.mat[i] <- norm(as.numeric(good.values[i,]) - xbar, type = "2")
}
div.mat <- rep(0, dim(good.values)[1]-1)
for (i in 1:dim(good.values)[1]-1){
  div.mat[i] <- good.values[i+2] / good.values[i+1]
}

plot(div.mat)
```



We can see the faster convergence rate (only requires 5 iterations) is quadratic, as expected from Newton method.