MATH319 – Assignment 3

Pierre Visconti

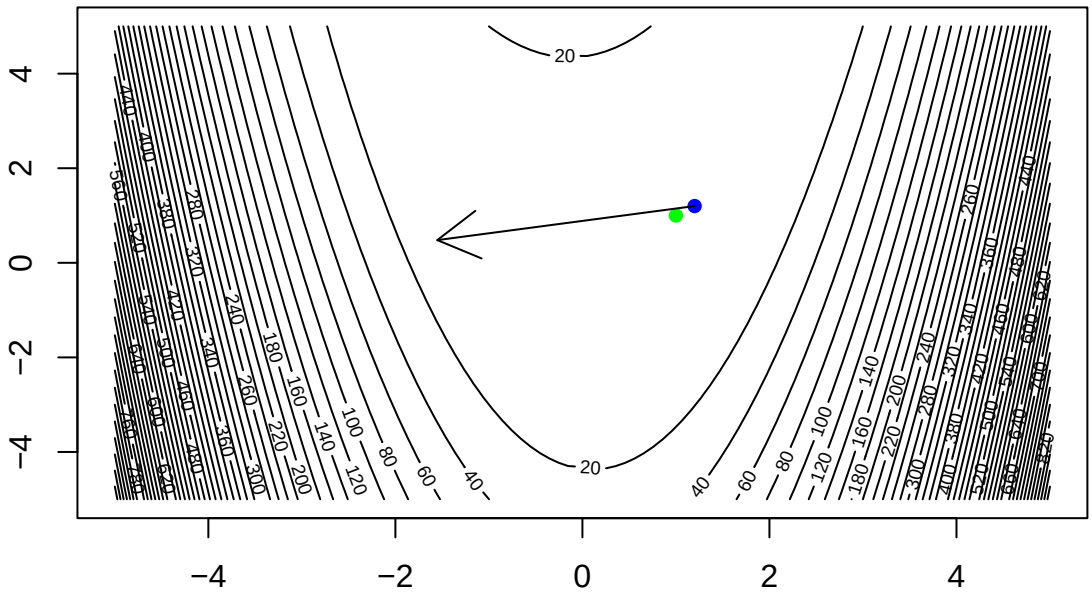
January 31, 2024

Part (a): Creating functions

```r
# rosenbrock function
rosenbrock <- function(x1, x2) {
  out <- (x2-x1^2)^2 + (1-x1)^2
  return(out)
}

# compute the gradient
grad <- function(x1, x2) {
  c(-4*x1*(x2-x1^2)-2*(1-x1), 2*(x2-x1^2))
}

# compute the hess
hess <- function(x1, x2) {
  matrix(c(-4*x2+12*x1^2+2, -4*x1, -4*x1, 2), 2, 2)
}
```

Part (b): Contour plot

```r
# make a contour plot
x1 <- seq(-5,5, length.out=100)
x2 <- x1
z <- outer(x1, x2,FUN=rosenbrock)
plot_contour <- contour(x1, x2, z, nlevels=50, main="Contour Plot of Rosenbrock Function")

# plot optimal solution
points(1,1, col="green", pch=16)

# plot initial point x0
x0 = c(1.2, 1.2)
points(x0[1], x0[2], col="blue", pch=16)

# compute and plot steepest descent for x0
grad_x0 <- grad(x0[1], x0[2])
arrows(x0[1], x0[2], -grad_x0[1], -grad_x0[2])
```



Part (c): Computing Hessian, eigenvalues, and Newton direction

```
# compute hessian at x0
hess_x0 = hess(x0[1], x0[2])
hess_x0
```

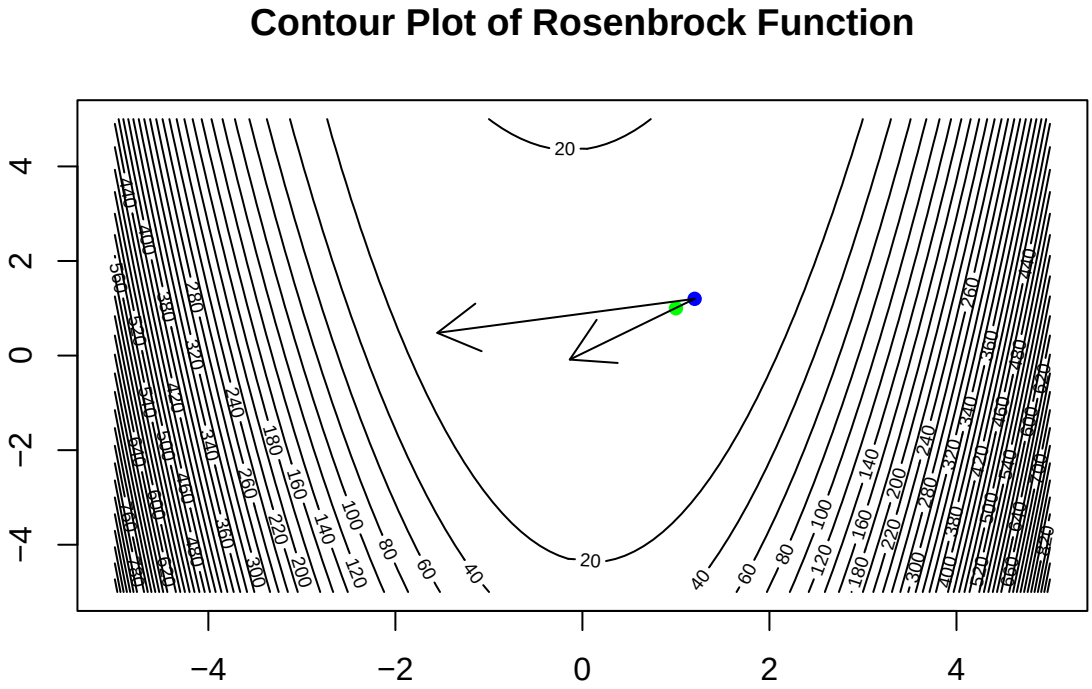
```
##       [,1] [,2]
## [1,] 14.48 -4.8
## [2,] -4.80  2.0
```

```
# compute eigenvalues for hessian at x0
eigen(hess_x0)
```

```
## eigen() decomposition
## $values
## [1] 16.1125853  0.3674147
##
## $vectors
##            [,1]       [,2]
## [1,] -0.9467376 -0.3220062
## [2,]  0.3220062 -0.9467376
```

```
# make a contour plot
x1 <- seq(-5,5, length.out=100)
x2 <- x1
z <- outer(x1, x2,FUN=rosenbrock)
plot_contour <- contour(x1, x2, z, nlevels=50, main="Contour Plot of Rosenbrock Function")

# plot optimal solution
points(1,1, col="green", pch=16)

# plot initial point x0
x0 = c(1.2, 1.2)
points(x0[1], x0[2], col="blue", pch=16)
```

```r
# plot steepest descent for x0
grad_x0 <- grad(x0[1], x0[2])
arrows(x0[1], x0[2], -grad_x0[1], -grad_x0[2])

# compute Newton direction for x0 and plot it
newt_x0 = -solve(hess_x0) %*% grad_x0
arrows(x0[1], x0[2], newt_x0[1], newt_x0[2])
```

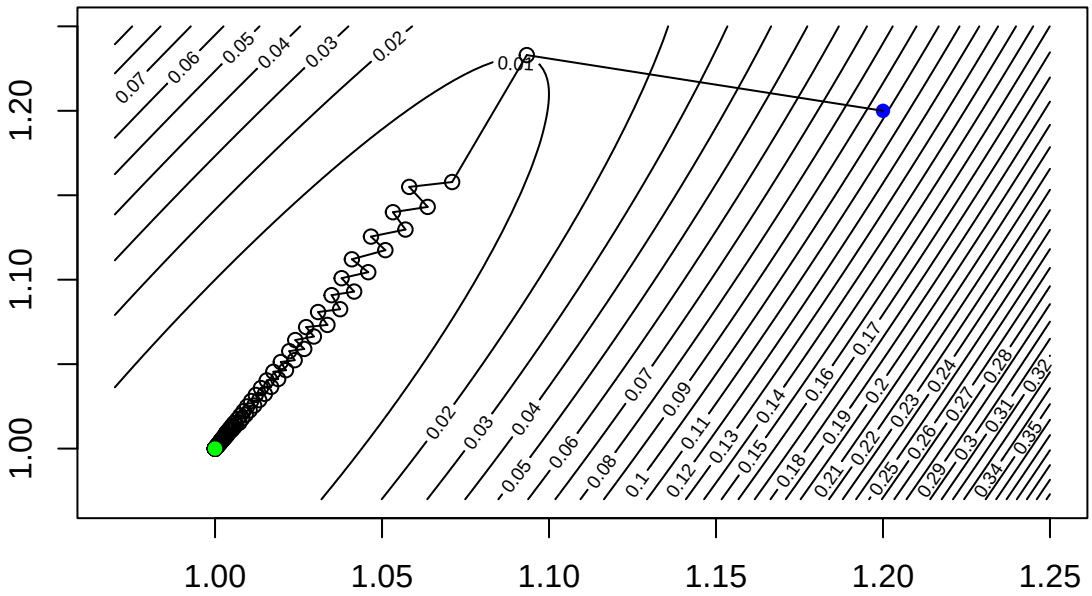


Note that the hessian at x0 is PD therefore it is convex and has a global minimizer p_k. Therefore Newton direction is well defined and the descent direction is given by p_k= -(hess)^(-1) * grad. The Newton direction looks better for minimizing f, for two reasons. First it is pointing more towards the optimal solution, and second, it has a smaller magnitude meaning that it will take less iterations to choose a desired step length.

Part (d): Full algorithm implementation

```r
# make a contour plot
x1 <- seq(0.97,1.25, length.out=100)
x2 <- x1
z <- outer(x1, x2,FUN=rosenbrock)
plot_contour <- contour(x1, x2, z, nlevels=50, main="Contour Plot of Rosenbrock Function")

# plot initial point x0
x0 = c(1.2, 1.2)
points(x0[1], x0[2], col="blue", pch=16)


count <- 0

# initialize variables
c1 <- 0.4
rho <- 0.8
xk <- c(1.2, 1.2)
```

```
grad_xk <- grad(xk[1], xk[2])
norm_grad <- norm(c(grad_xk[1],grad_xk[2]), type="2")

while(count < 500 && norm_grad > 0.000001) {
  a <- 1
  # choose a descent direction
  pk <- -grad(xk[1], xk[2])
  # initialize step length variables
  x_ap <- xk + a*pk
  f.x_ap <- rosenbrock(x_ap[1], x_ap[2])
  f.x <- rosenbrock(xk[1], xk[2])
  grad.x_cap <- c1*a*t(grad(xk[1], xk[2]))%*%pk
  # compute step length based on first Wolfe condition
  while (f.x_ap > f.x + grad.x_cap) {
    # update step length
    a <- rho*a
    # update inequality
    x_ap <- xk + a*pk
    f.x_ap <- rosenbrock(x_ap[1], x_ap[2])
    grad.x_cap <- c1*a*t(grad(xk[1], xk[2]))%*%pk
  }
  # update ak and norm
  xk_1 <- xk
  xk <- xk + a*pk
  grad_xk <- grad(xk[1], xk[2])
  norm_grad <- norm(c(grad_xk[1],grad_xk[2]), type="2")
  count <-  count + 1
  # plot xk
  points(xk[1], xk[2])
  segments(xk_1[1], xk_1[2], xk[1], xk[2])
}

# plot optimal solution
points(1,1, col="green", pch=16)
```




```
print(count)
```

```
## [1] 194
```

```
print(xk)
```

```
## [1] 1.000001 1.000002
```

Part (e): Performance evaluation

I believe that the algorithm performed well, considering it only took 194 iterations to obtain a final x_k value of (1.000001, 1.000002), which is close to the optimal solution of (1,1). Notably, lowering the norm exit condition to 0.0001 from 0.000001 lowered the iterations from 194 to 116 while having a final x_k value of 1.000084 and 1.000191 which could still be acceptable depending on your requirements. That being said, there is definently room for improvement as it wasted a lot of time zigzagging while barely moving in the direction of the optimal solution.