MATH319Final

```
suppressWarnings(library(reticulate))
```

Importing python code

```
auto_diff <- reticulate::import("forward_auto_diff")
```

Defining parameters

```
x0 <- c(1.2, 1.2)
c1 <- 0.4
rho <- 0.8
tol <- 0.000001
iter <- 500
```

Steepest descent algorithm with forward auto diff

```
# steepest descent function
steepest_descent <- function(func, x0, identity) {
  # plot initial point x0
  points(x0[1], x0[2], col="blue", pch=16)

  # initialize variables
  xk <- x0
  ####### computing gradient from auto_diff ########
  var1 <- auto_diff$Var(xk[1], 0)
  var2 <- auto_diff$Var(xk[2], 0)
  if (identity == 1) {grad <- auto_diff$compute_grad(c(var1, var2))}
  else {grad <- auto_diff$compute_grad_a4(c(var1, var2))}
  ##################################################
  norm_g <- norm(c(grad[1], grad[2]), type="2")
  count <- 0

  # algorithm logic
  while(count < iter && norm_g > tol) {
    a <- 1
    pk <- -grad
    # initialize step length variables
    x_ap <- xk + a*pk
    wolfe = func(x_ap[1], x_ap[2]) <
            (func(xk[1], xk[2]) + c1*a*t(grad)%*%pk)
```

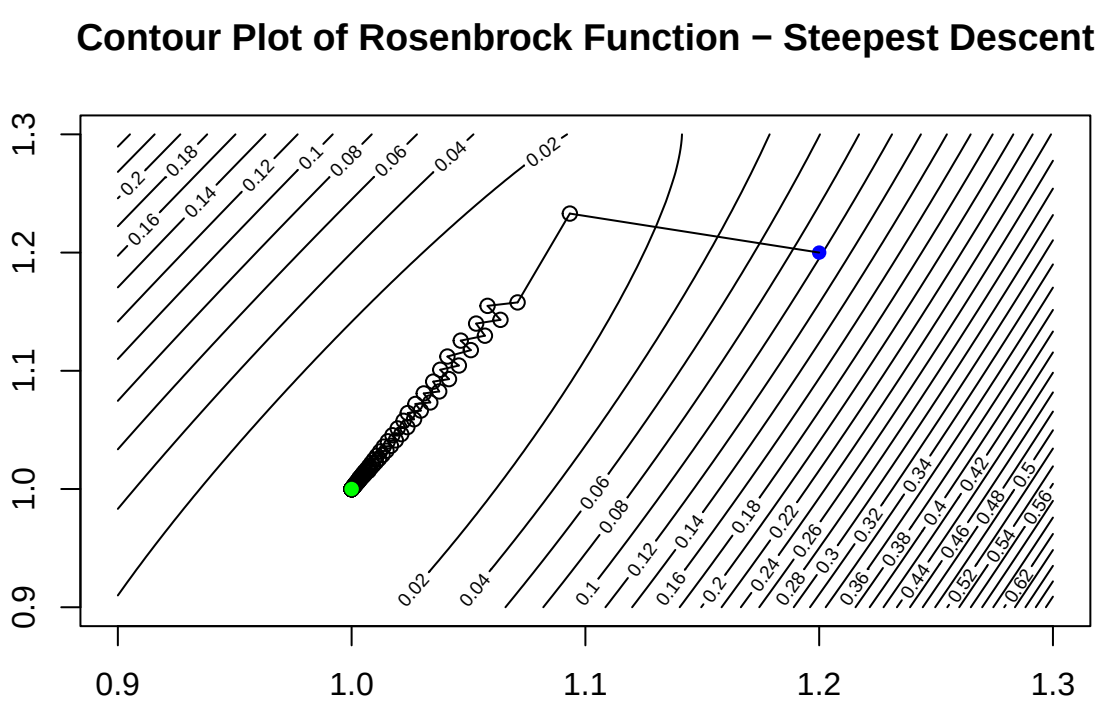
```
    # compute step length based on first Wolfe condition
    while (!wolfe) {
      # update step length
      a <- rho*a
      # update wolfe condition
      x_ap <- xk + a*pk
      wolfe = func(x_ap[1], x_ap[2]) <
              (func(xk[1], xk[2]) + c1*a*t(grad)%*%pk)
    }
    # update iterate and norm
    xk_o <- xk
    xk <- xk + a*pk
    ####### computing gradient from auto_diff ########
    var1 <- auto_diff$Var(xk[1], 0)
    var2 <- auto_diff$Var(xk[2], 0)
    if (identity == 1) {grad <- auto_diff$compute_grad(c(var1, var2))}
    else {grad <- auto_diff$compute_grad_a4(c(var1, var2))}
    ##################################################
    norm_g <- norm(c(grad[1], grad[2]), type="2")
    count <-  count + 1
    # plot iterate
    points(xk[1], xk[2])
    segments(xk_o[1], xk_o[2], xk[1], xk[2])
  }

  cat("Iterations: ", count, "\n")
  cat("Optimal value: ", xk)
}
```

Applying steepest descent with forward auto diff on rosenbrock function

```
# make a contour plot
x1 <- seq(0.9, 1.3, length.out=100)
x2 <- x1
g <- Vectorize(auto_diff$funct)
z <- outer(x1, x2,FUN=g)
plot_contour <- contour(x1, x2, z, nlevels=50,
                        main="Contour Plot of Rosenbrock Function - Steepest Descent")
steepest_descent(g, x0, 1)
```

```
## Iterations:  194
## Optimal value:  1.000001 1.000002
```

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



Applying steepest descent with forward auto diff on function A4

```
# make a contour plot
x1 <- seq(-5, 15, length.out=100)
x2 <- x1
g <- Vectorize(auto_diff$funct_a4)
z <- outer(x1, x2,FUN=g)
plot_contour <- contour(x1, x2, z, nlevels=50,
                        main="Contour Plot of Function A4 - Steepest Descent")

# perform steepest descent with various starting points
steepest_descent(g, c(0, 6.1), 2)
```

```
## Iterations:  198
## Optimal value:  -3.141592 12.275
```

```
steepest_descent(g, c(0, 5.9), 2)
```

```
## Iterations:  50
## Optimal value:  3.141593 2.275
```

```r
steepest_descent(g, c(6.5, 13), 2)
```

```
## Iterations:  49
## Optimal value:  9.424778 2.475
```

```r
# find the function value for each optimal solution found
g(-3.141592, 12.275)
```

```
## [1] 0.3978874
```

```r
g(3.141593, 2.275)
```

```
## [1] 0.3978874
```

```r
g(9.424778, 2.475)
```

```
## [1] 0.3978874
```

```r
points(-10,-10) # used to display plot as final output
```