Statistical Computing - Assessed Coursework 1

Conor Newton

Contents

1 The Package

I have created a package that provides functions for both gradient descent and stochastic gradient descent.

The package and all of the files for this project can be found here on github:

https://www.github.com/conornewton/sc1-optimization

1.1 Installation

This package can be installed directly from github using the following command in an R shell if `devtools` is installed

```
devtools::install_github("conornewton/sc1-optimization")
```

This package has no required dependencies.

To load the package in R, use the following command

```
library(sc1optimization)
```

Notice that there is no hyphen in the package name here.

1.2 Documentation

The documentation for this package is generated automatically from the source files using `roxygen2`.

The package exports two functions, `grad_descent` and `stoc_grad_descent`. The documentation for them can be accessed from the shell using the `?grad_descent` and `?stoc_grad_descent` commands.

Alternatively, the documentation can be accessed in a pdf here:

https://www.github.com/conornewton/sc1-optimization/doc/man.pdf

1.3 Testing

This package uses `testthat` for unit testing. Both the `gradDescent` and `stocGradDescent` have unit tests to ensure that they work for a variety of edge cases. These test are found in the `tests/testthat` directory. If `testthat` is installed locally, the package will be tested automatically on installation.

After installation, the tests can be run manually using the following command in the R shell

```
devtools::test("sc1-optimization")
```

2 Gradient Descent

This focuses on the methodology of my gradient descent implementation. The relevant source code and tests can be found at the following urls

- https://github.com/conornewton/sc1-optimization/blob/master/R/gradDescent.R
- https://github.com/conornewton/sc1-optimization/blob/master/tests/testthat/testGradDescent.R

2.1 Introduction

Gradient descent methods attempt to find a local minimum of a differentiable functions f by moving along the curve in the opposite direction to the gradient at each point. Since the gradient gives the direction of the "fastest increase" moving in the opposite direction should gives us the fastest decrease. Gradient descent is performed by iterating the following

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \gamma_n \nabla f(\mathbf{x}_{n-1})$$

Here γ_n is the step-size. This determines how much we should move in the opposite direction of gradient in each step. For simplicity, it can be choose to be constant. More effective gradient descent approaches use a systematic way of determining γ_n on each iteration, such as a line search.

2.2 The Gradient

For our gradient descent implementation, we have the option of passing as a parameter the gradient of the function we would like to minimize otherwise it will estimate the gradient at each point using finite differencing.

Finite differencing at a point $\mathbf{x}$ works by finding the gradient of the line passing through two points on each side and close to $\mathbf{x}$. Each component of the gradient is computed by the following

$$(\nabla f(\mathbf{x}))^{(i)} \approx \frac{f(\mathbf{x} + \Delta \mathbf{e}_i) - f(\mathbf{x} - \Delta \mathbf{e}_i)}{2\Delta}$$

where Δ is a small constant.

The gradient is also used to decide when we have found a local minimum. If the magnitude of the gradient is small, then we are a point with little curvature and therefore close to a local minimum. For our gradient descent implementation, the user can provide a tolerance for the magnitude of the gradient to decide when to terminate the search.

2.3 Step Sizes

In this package, we have the choice of using the following three different approaches to calculate step size

- Constant step size

- Line Search
- Barzilai-Borwein

All three of these approaches share the same guarantee that gradient descent will converge to a local minimum if f is convex and its gradient ∇f is Lipschitz.

2.3.1 Constant Step Size

Choosing a constant step size is the most basic approach to gradient descent. For every iteration, our estimate for the minimum $\mathbf{x}$ moves by the same distance in the opposite direction of the gradient. This can often lead to overshooting/undershooting the minimum in that direction, therefore this approach can take long to converge. The next two methods both give faster convergence.

2.3.2 Backtracking Line Search

Line search selects the step size γ_n that satisfies the following

$$\gamma_n = \underset{\gamma}{\operatorname{argmin}} f(\mathbf{x}_{n-1} - \gamma \nabla f(\mathbf{x}))$$

This is finding finding the argument γ that minimises f on the line $(\mathbf{x}_{n-1} - \gamma \nabla f(\mathbf{x}))$.

A backtracking approach can be used to quickly estimate this line search step size. This works by iterating

$$\alpha_n = \tau \alpha_{n-1}$$

where α_n is the step size and $\tau \in (0, 1)$ until the following is satisfied

$$f(\mathbf{x}) - f(\mathbf{x} - \alpha_j \nabla f(\mathbf{x})) \geq c \alpha_j ||\nabla f(\mathbf{x})||^2$$

for some $c \in (0, 1)$. Essentially we are just looking for a sufficient drop in the objective function in the direction of $\nabla f(\mathbf{x})$.

2.3.3 Barzilai-Borwein Method

A third approach to calculating the step size is the Barzilai-Borwein method. This approach takes into consideration the previous estimate $\mathbf{x}_{n-1}$. The step size is calculated as follows

$$\gamma_n = \frac{(\mathbf{x} - \mathbf{x}_{n-1}) \cdot (\nabla f(\mathbf{x}_n) - \nabla f(\mathbf{x}_{n-1}))}{||\nabla f(\mathbf{x}_n) - \nabla f(\mathbf{x}_{n-1})||^2}$$

2.4 Usage

Here we give some examples of how we can use the `grad_descent` function.

Firstly, we can use `grad_descent` to estimate the argument of the local minimum of the Rosenbrock function using a constant step size. Our initial guess is the point $(0, 0)$ and our optimization will not exceed 100000 iterations.

```r
# Rosenbrock function
f <- function(x) (1 - x[1])^2 + 100 * (x[2] - x[1]^2)^2
grad_descent(f, c(0, 0), n = 100000, step_method = 0.01)
```

This can be estimated much quicker if we choose a different step methods.

```r
# Uses Barzilai-Borwein
f <- function(x) (1 - x[1])^2 + 100 * (x[2] - x[1]^2)^2
grad_descent(f, c(0, 0), n = 100000, step_method = "BB")
```

```r
# Uses Backtracking Line Search
f <- function(x) (1 - x[1])^2 + 100 * (x[2] - x[1]^2)^2
grad_descent(f, c(0, 0), n = 100000, step_method = "BLS")
```

We can give our `grad_descent` function an explicit gradient of f and choose a tolerance for our gradient as follows

```r
f <- function(x) x[1]^2 + x[2]^2 + 3 * x[1] + 2 * x[2] + 9
grad_f <- function(x) c(2 * x[1] + 3, 2 * x[2] + 2)
grad_descent(f, c(0, 1), grad_f, tol = 1e-4)
```

This summarises pretty much all of the functionality of the `grad_descent` function.

3 Stochastic Gradient Descent

This focuses on the methodology of my gradient descent implementation. The relevant source code and tests can be found at the following urls

- https://github.com/conornewton/sc1-optimization/blob/master/R/stochasticGradDescent.R
- https://github.com/conornewton/sc1-optimization/blob/master/tests/testthat/testStocGradDescent.R

3.1 Introduction

Stochastic gradient descent is a gradient descent method that is commonly used to find minimum of a function that depends on a large data set.

Instead, if we can break up the objective function into n pieces

$$Q(\mathbf{x}) = \sum_{i=1}^{n} Q_i(\mathbf{x})$$

where each Q_i only involves using the i'th data point, we can then apply stochastic gradient descent. Functions of this form are common in machine learning, such as mean square error, log-likelihood etc.

Stochastic gradient descent can be performed by iterating the following

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \gamma \nabla Q_i(\mathbf{x})$$

In each iteration we are only considering a single data point, but by the end of the computation all of the data points have been involved in learning $\mathbf{x}$. For our simple implementation we have set γ to be constant.

For the `stoc_grad_descent` function, the gradient is estimated using finite differencing.

3.2 Usage

We can use `stoc_grad_descent` to estimate the argument that minimises the mean square error as follows

```
# Summand of the objective function
f <- function(w, x, y) (sum(w * c(1, x)) - y)^2

# Generating a data set which will appear in the summand
y <- mapply(function(x1, x2) sum(c(20, 1) * c(x1, x2)) + 2, 1:100, 1:100)
data <- data.frame(1:100, 1:100, y)

stoc_grad_descent(f, data, c(0, 0, 0))
```