Optimization

2023-11-28

Numerical Optimization

This portfolio will focus on minimizing a continous function (which is equivalent to maximizing the negative). In general, optimization problems are of the form

$$\arg\min_x f(x) \text{ subject to } g_i(x) \leq 0 \text{ for } i \in \{1, ..., m\} \text{ and } h_j(x) = 0 \text{ for } j \in \{1, ..., p\}$$

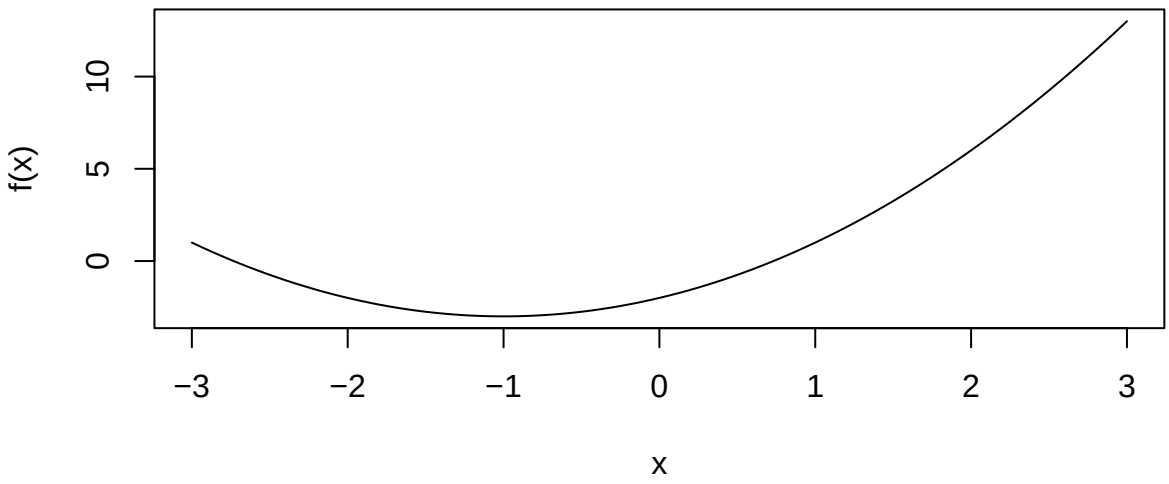
where we are minimizing a function $f : \mathbb{R}^n \to \mathbb{R}$ } subject to some inequality and equality constraints. Optimization problems that are discrete and/or have constraints are more difficult to solve than continuous and/or unconstrained problems.

One-dimensional optimization

`optimize` function

Base R has the `optimize` function, which only works on one dimension. `optimize` attempts to find a minimum using a golden section search algorithm, which works well for uni-modal functions where the minimum is within the specified interval.

```
f <- function(x) x^2 + 2*x - 2
curve(f, from = -3, to = 3)
```

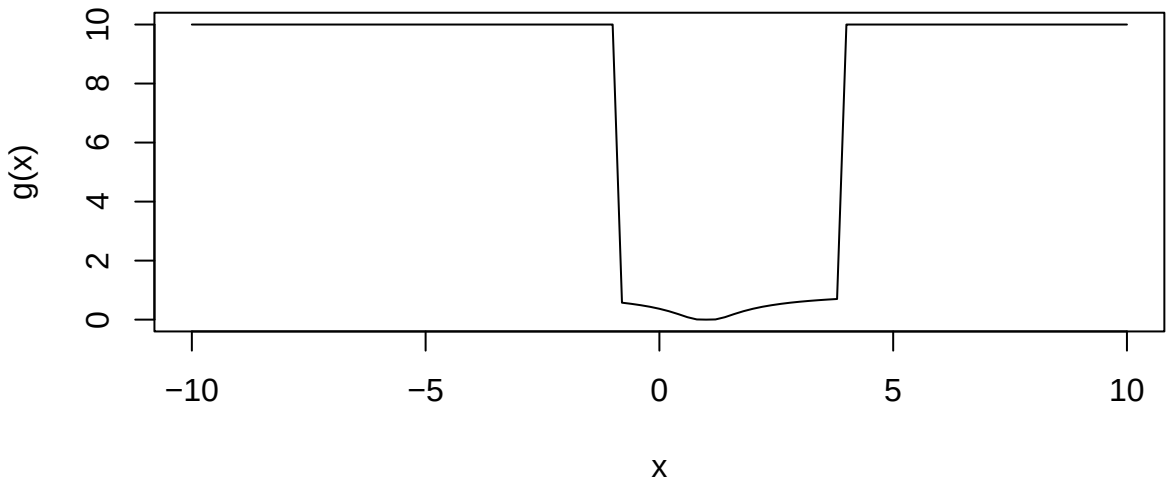


```r
optimize(f, interval = c(-3, 3))
```

```
## $minimum
## [1] -1
##
## $objective
## [1] -3
```

The golden section search algorithm can struggle to find global minima, even with relatively simple functions.

```r
g  <- function(x) ifelse(x > -1, ifelse(x < 4, exp(-1/abs(x - 1)), 10), 10)
curve(g, from = -10, to = 10)
```




```r
optimize(g, interval = c(-4, 20))
```

```
## $minimum
## [1] 19.99995
##
## $objective
## [1] 10
```

```r
optimize(g, interval = c(-8, 20))
```

```
## $minimum
## [1] 1.000343
##
## $objective
## [1] 0
```

Multi-dimensional optimization

In multi-dimensional optimization problems, there are three general categories of algorithms:

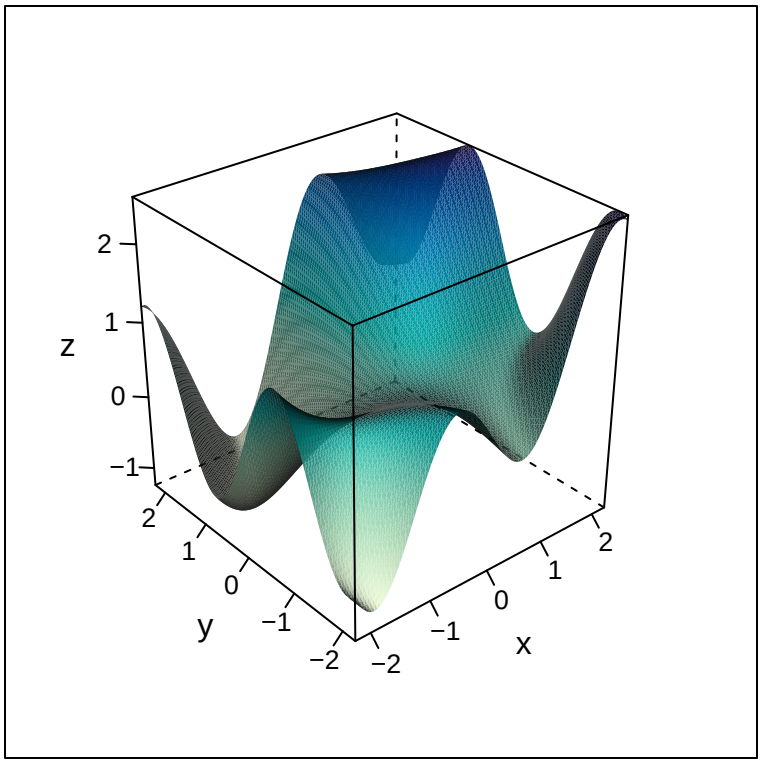
- Simplex methods, which use the value of the function
- Gradient methods, which use the value of the function and its gradient
- Newton methods, which use the value, the gradient, and the Hessian (or an approximation)

In addition to these techniques, simulated annealing is a technique that aims to find the global minimum of a function, although it often fails to work in real-world settings.

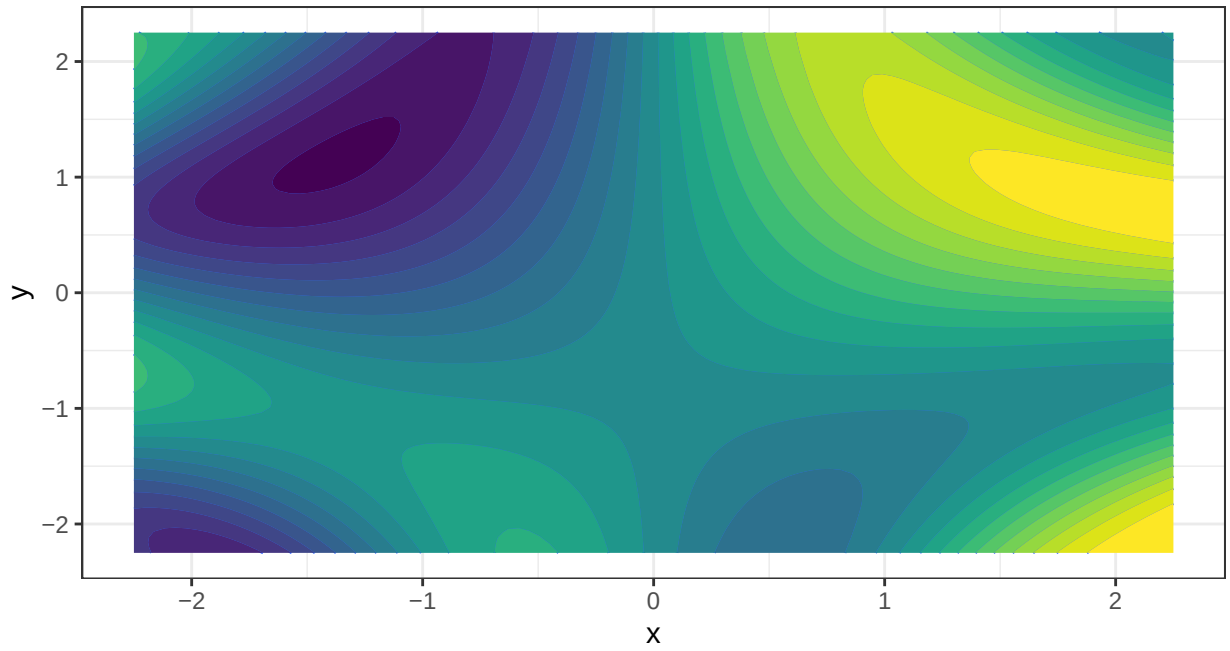
`optim` function

The `optim` function gives a choice of algorithms from the types of optimization listed above. We can apply several methods to the following example function:

```
library(lattice)
library(ggplot2)
multivar_f <- function(X){
  if(is.vector(X)) {
    X <- matrix(X, ncol = 2)
  }
  x1 <- X[, 1]
  x2 <- X[, 2]
  value <- (0.5*x1)^2 + sin(x1*x2) + cos(x1 - 1)
  return(value)
}
# Plot the function
x1 <- seq(-2.25, 2.25, length = 101)
x2 <- seq(-2.25, 2.25, length = 101)
X <- as.matrix(expand.grid(x1, x2))
colnames(X) <- c("x", "y")
z <- multivar_f(X)
df <- data.frame(X, z)
wireframe(z ~ x * y,
          data = df,
          shade = TRUE,
          scales = list(arrows = FALSE)
          )
```




```
ggplot(df, aes(x, y, z = z)) +
  geom_contour(binwidth = 0.2) +
  geom_contour_filled(binwidth = 0.2) +
  theme_bw() +
  theme(legend.position = "none")
```



The Nelder-Mead algorithm is a simplex method:

```
optim(par = c(0,0), multivar_f, method = "Nelder-Mead")
```

```
## $par
## [1] -1.380115  1.138140
##
## $value
## [1] -1.247638
##
## $counts
## function gradient
##       61       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Conjugate gradient is a gradient descent method that attempts to mitigate the zigzag behavior common in steepest descent:

```
optim(par = c(0,0), multivar_f, method = "CG")
```

```
## $par
## [1] -1.380060  1.138209
##
## $value
## [1] -1.247638
##
```

```
## $counts
## function gradient
##       69       29
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Traditional Newton methods require the inverse of the Hessian. Since the Hessian is a $n \times n$ dense matrix, computing the inverse can be computationally difficult in high dimensions. BFGS is a quasi-Newton method, meaning that it approximates the Hessian and avoids the computational complexity of matrix inversion.

```
optim(par = c(0,0), multivar_f, method = "BFGS")
```

```
## $par
## [1] -1.380061  1.138208
##
## $value
## [1] -1.247638
##
## $counts
## function gradient
##       19       10
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The three methods shown all find a similar optimum and run quickly, but BFGS requires far fewer calls to the function. Note that Nelder-Mead does not require any calculation of gradients, so although it requires the most calls to the function, it can be useful in situations where the gradient is difficult to compute.