Numeric Optimization

Stats 102A

Miles Chen

Department of Statistics

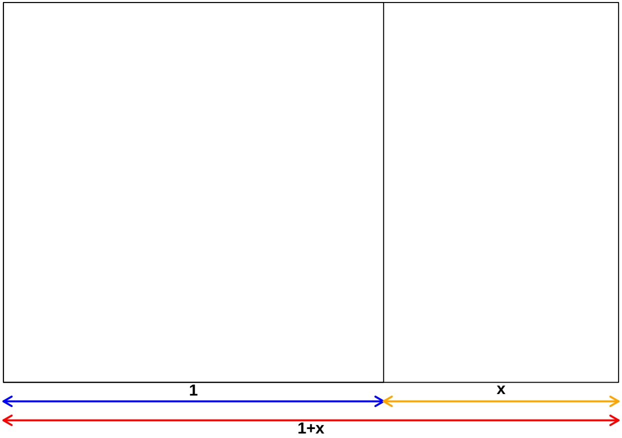
Week 8 Monday

UCLA

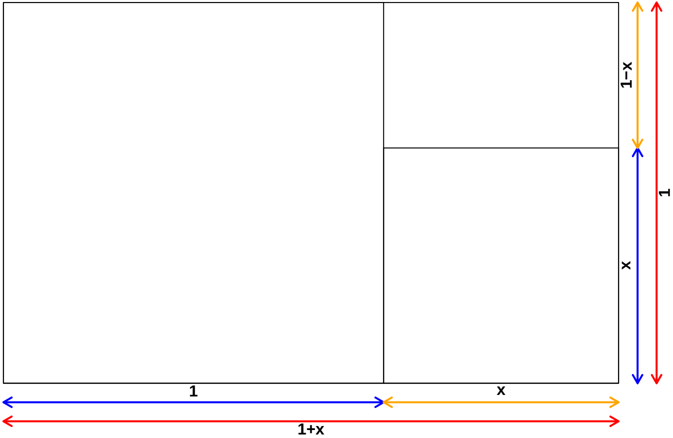
Section 1

Golden Section Search

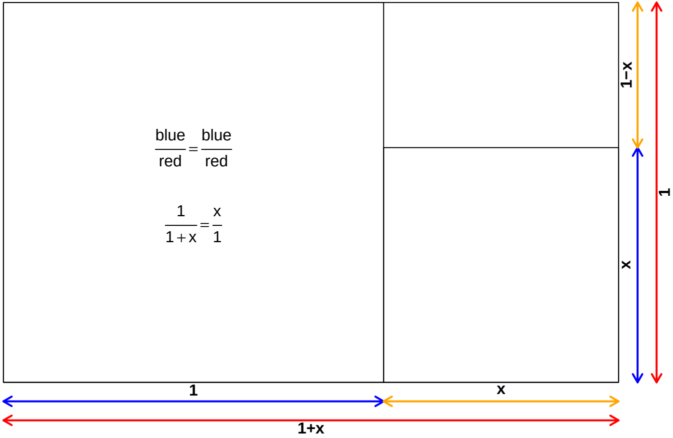
Golden Ratio



Golden Ratio



Golden Ratio



Golden Ratio

$$
\begin{aligned}
\frac{1}{1+x} &= x \\
1 &= x(1+x) \\
0 &= x^2 + x - 1 \\
x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-1 \pm \sqrt{1^2 + 4}}{2} \\
x &= \frac{-1 + \sqrt{5}}{2} \approx 0.618 \quad \text{(in the diagram x is a positive distance)}
\end{aligned}
$$

Golden Ratio

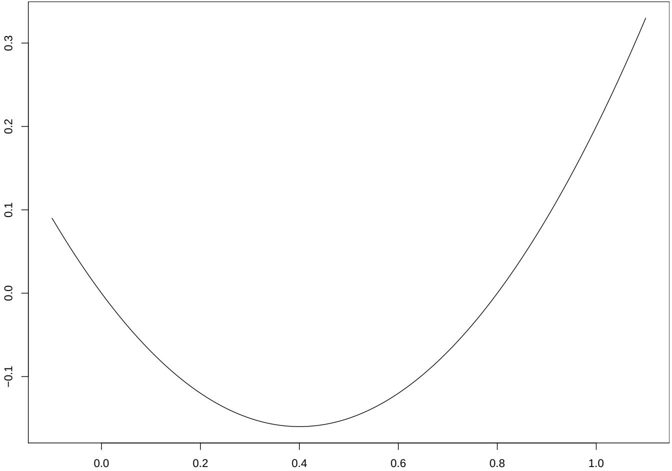
$$\frac{1}{1+x} = x$$
$$\frac{1}{1.618} \approx 0.618$$

Also:

$$
\begin{aligned}
0 &= x^2 + x - 1 \\
1 - x &= x^2 \\
1 - 0.618 &\approx 0.618^2 \\
0.382 &\approx 0.618^2
\end{aligned}
$$

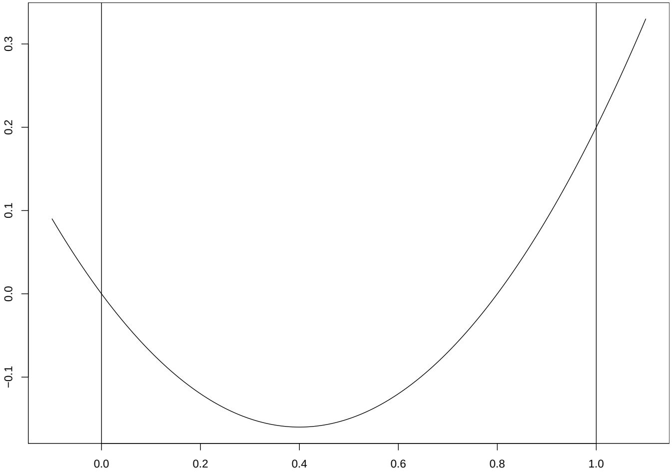
Optimization with Golden Section Search

Let's start with a simple function that we wish to minimize. $f(x) = x^2 - 0.8x$



Golden Section Search

We select an arbitrary interval with the assumption that the minimum value exists somewhere between the endpoints. I choose the lower endpoint to be 0 and the upper endpoint to be 1.

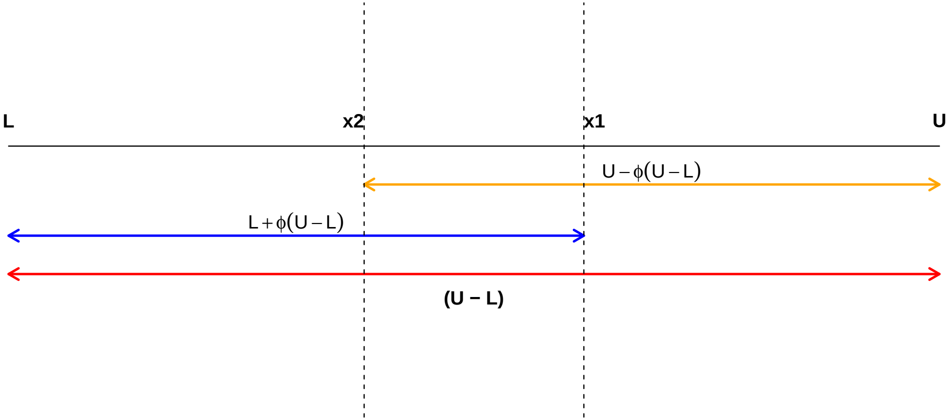


Golden Section Search

We evaluate the function at two internal points. Let d be the distance between endpoints, i.e. the width of the interval.

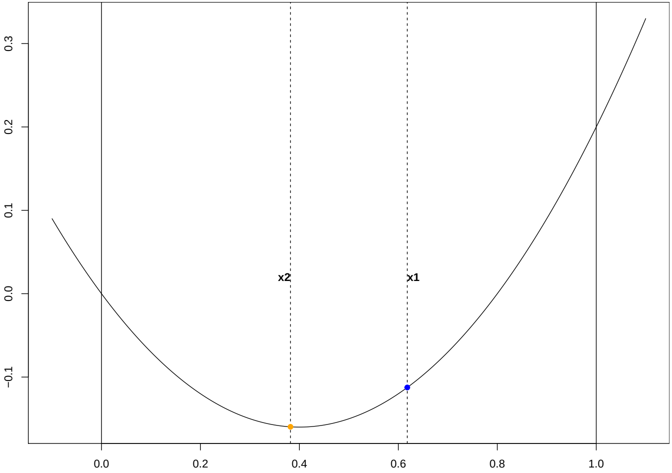
The internal points are

- $x_1 = l + \phi(u - l) \approx 0 + 0.618(1) \approx 0.618$
- $x_2 = u - \phi(u - l) \approx 1 - 0.618(1) \approx 0.382$



Golden Section Search

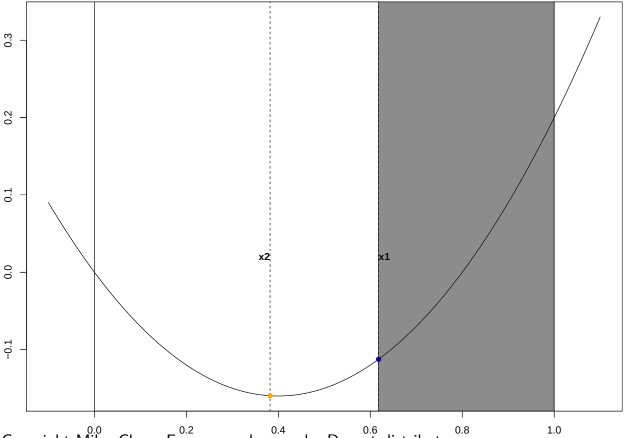
We evaluate $f(x_1)$ and $f(x_2)$. We find that $f(x_1) > f(x_2)$.



Golden Section Search

Because $f(x_1) > f(x_2)$, we know that the minimum must lie in between l and x_1. The minimum value cannot be to the right of x_1 because we know a value (x_2) exists between l and x_1 that produces a smaller value.

We eliminate the area to the right of x_1 and we set the boundaries of our search interval to be from l to x_1. That is to say, x_1 becomes the new upper bound u of the search interval.



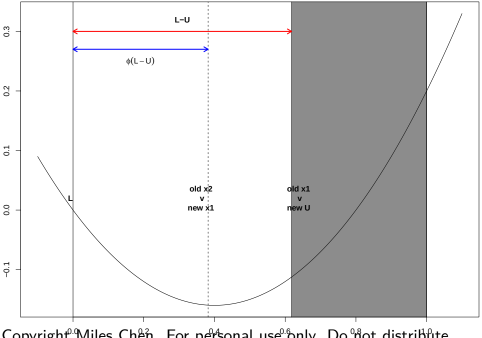
Golden Section Search

The old x_1 becomes the new upper boundary u. The lower boundary l remains.

The genius of the golden section search is that we used the golden ratio to determine the inner points. So x_2 which was located at ≈ 0.382 becomes the new x_1.

$$\text{new } x_1 = l + \phi(\text{new } u - l) \approx 0 + 0.618(0.618 - 0) \approx 0.618^2 \approx 0.382 = \text{old } x_2$$

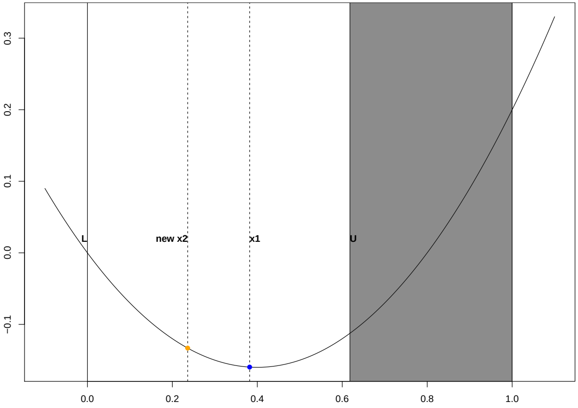
We reuse the old x_2 and $f(x_2)$ values. We do not need to recalculate these numbers as they have already been calculated before. This saves a bit of computational power.



Golden Section Search

We continue the process. We must locate a new x_2 value and then evaluate the function at x_2.

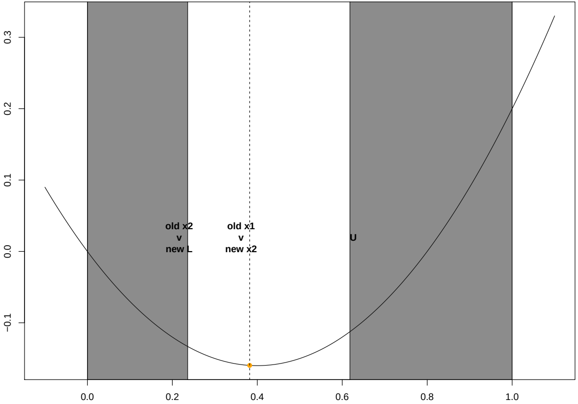
$$x_2 = u - \phi(u - l) \approx 0.618 - 0.618(0.618 - 0) \approx 0.236$$



Golden Section Search

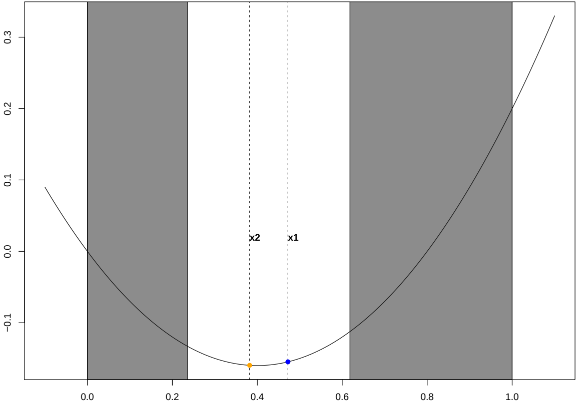
Because $f(x_1) < f(x_2)$, we know that the minimum must lie in between x_2 and U. The minimum value cannot be to the left of x_2 because we know a value (x_1) exists between x_2 and u that produces a smaller value.

We eliminate the area to the left of x_2.

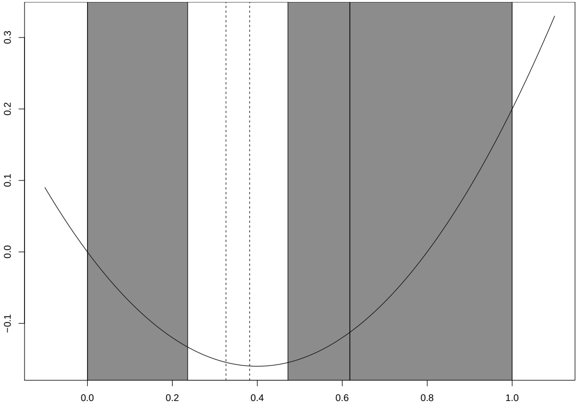


Golden Section Search

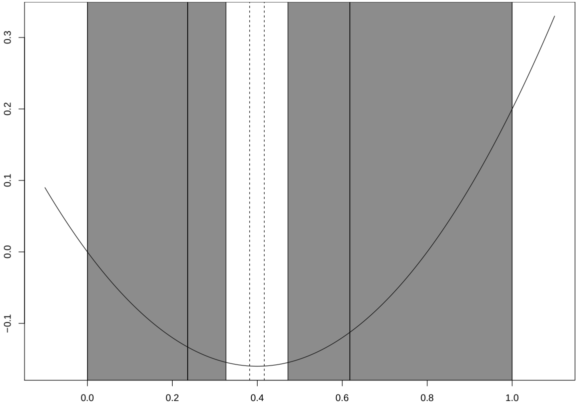
- old x_2 becomes the new lower bound l
- old x_1 becomes the new interior point x_2
- We calculate a new x_1 and evaluate $f(x_1)$



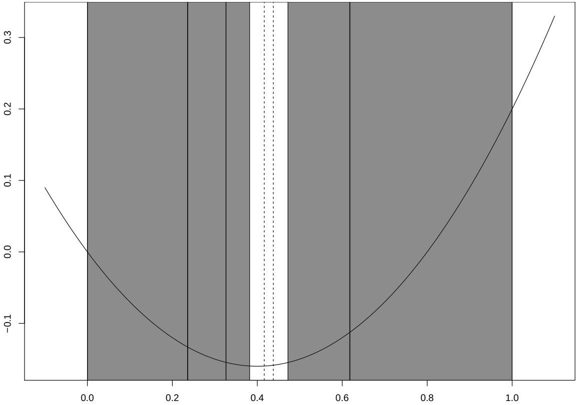
Golden Section Search



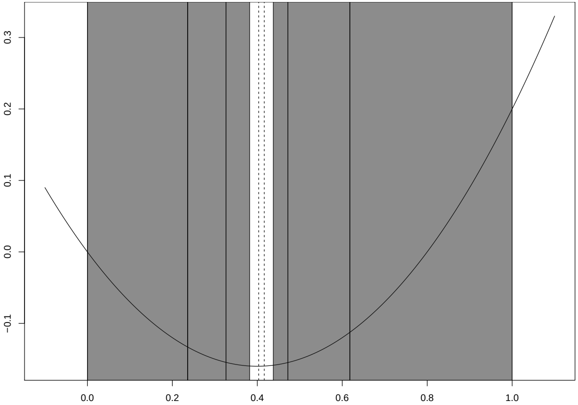
Golden Section Search



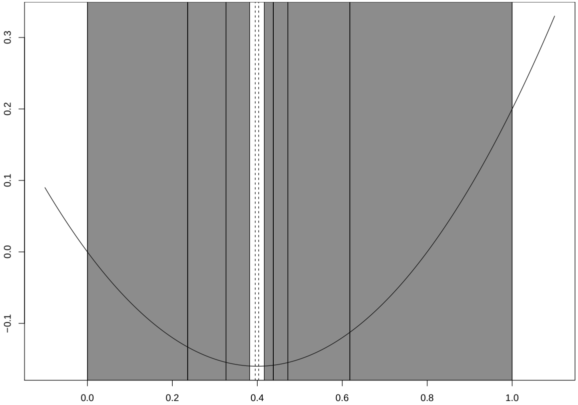
Golden Section Search



Golden Section Search



Golden Section Search



Golden Section Search

Golden section search is very robust.

- If a minimum lies within the interval, it will find the minimum.
- If the minimum lies on the interval boundary, it will converge to the boundary point.
- If multiple local minimum exist within the interval, the algorithm will converge to one of the local minima

Termination condition:

- multiple termination conditions are possible. A common choice is when the difference between the upper and lower bounds of the interval is less than some arbitrary tolerance value.

```r
##### A modification of code provided by Eric Cai
golden <- function(f, lower, upper, tolerance = 1e-5) {
  phi <- (-1 + sqrt(5))/2
  ## Use the golden ratio to find the initial test points
  x1 <- lower + phi * (upper - lower)
  x2 <- upper - phi * (upper - lower)
  ## the arrangement of points is:
  ## lower ----- x2 --- x1 ----- upper
  ### Evaluate the function at the test points
  f1 <- f(x1)
  f2 <- f(x2)
  iterations <- 0
  while (abs(upper - lower) > tolerance) {
    if (f2 > f1) {
      # the minimum is to the right of x2
      lower <- x2  # x2 becomes the new lower bound
      x2 <- x1     # x1 becomes the new x2
      f2 <- f1     # reuse f(x1). now becomes f(x2)
      x1 <- lower + phi * (upper - lower) # calculate new x1
      f1 <- f(x1)  # calculate new f(x1)
    } else {
      # then the minimum is to the left of x1
      upper <- x1  # x1 becomes the new upper bound
      x1 <- x2     # x2 becomes the new x1
      f1 <- f2     # reuse f(x1). now becomes f(x2)
      x2 <- upper - phi * (upper - lower) # calculate new x2
      f2 <- f(x2)  # calculate new f(x2)
    }
    iterations <- iterations + 1
  }
  cat("Converged in:", iterations, "iterations.\n") # print
  (lower + upper)/2 # the returned value is the midpoint of the bounds
}
```

Convergence

```r
f <- function(x) { x ^ 2 - 0.8 * x } # true minimum at 0.4
golden(f, 0, 1, tol = 1e-5)
```

```
## Converged in: 24 iterations.
```

```
## [1] 0.3999999
```

```r
golden(f, 0, 1, tol = 1e-6)
```

```
## Converged in: 29 iterations.
```

```
## [1] 0.4000001
```

```r
golden(f, 0, 1, tol = 1e-7)
```

```
## Converged in: 34 iterations.
```

```
## [1] 0.4
```

```r
golden(f, 0, 1, tol = 1e-8)
```

```
## Converged in: 39 iterations.
```

```
## [1] 0.4
```

Convergence

The width of each iteration shrinks by a factor of $\phi \approx .618$ compared to the previous iteration.

After two additional iterations, the width of the interval will be $\phi^2 \approx 0.382$ of the original interval.

```r
phi <- (-1 + sqrt(5))/2
phi ^ 5
```

```
## [1] 0.09016994
```

If you want your interval to shrink by a factor of 10 (to produce an additional decimal place of precision), it will require about 5 additional iterations.

Section 2

Coordinate Descent

Optimization via grid search

Univariate optimization is relatively easy. Even without an algorithm, we can approximate a minimum via "brute force" by performing a gird search.

A grid search is done by evaluating the function at many locations. This is effectively what we do when we graph a function on the computer. In the following code, I evaluate $f(x)$ at 6001 locations between -3 and 3 and identify which value of x produced the smallest value of $f(x)$.

```
x <- seq(-3, 3, by = 0.001)
length(x)
```

```
## [1] 6001
```

```
fx <- x^2 - 0.8 * x
which.min(fx) # index of the minimum value of f
```

```
## [1] 3401
```

```
x[which.min(fx)] # the corresponding x
```

```
## [1] 0.4
```

Multivariate optimization

Grid search can be extended to multivariate functions as well. The number of locations to evaluate, however, grows exponentially.

Let's say we have a function of two variables: $f(x, y)$.

We may want to evaluate many points inside a square covering from (-3, -3) to (3, 3). If the locations are spaced out by 0.001, there are about 6000 locations in each direction to evaluate.

In total, there are approximately \$6000ˆ2 = \$ 36 million locations to evaluate.

As the number of dimensions grow, the number of locations grow exponentially.

Coordinate Descent

The coordinate descent algorithm is an optimization algorithm that searches for the minimum of a multivariate function by performing univariate minimization in one direction at a time.

We achieve this by selecting one of the variables to minimize. All other variables in the function are held constant. This effectively reduces the multivariate function to a univariate one. We then perform minimization on the univariate function. Once we determine the minimum in this direction, we perform univariate minimization for the next variable.

Coordinate descent example

View the code example