# HUDM 6026

Univariate Optimization

## Introduction to Optimization

In statistics we are often concerned with estimating the parameters of a statistical model.

When we do that, we seek estimates which maximize some function (such as the likelihood) or minimize some function (such as the residual sum of squares).

When analytic expressions are available (such as the normal equations for the multiple regression problem) we should use them!

When they are not (or are too time consuming to work out) we can turn to numerical methods for optimization.

## Introduction to Optimization

We begin by discussing optimization of a univariate function (i.e., a function that has its domain in the real number line and its range in the real number line).

$$f : \Re_1 \rightarrow \Re_1$$

We discuss four method for univariate optimization. Although these methods are rather simple (because they are only searching over one dimension), they provide a good background for moving to higher dimensional methods.

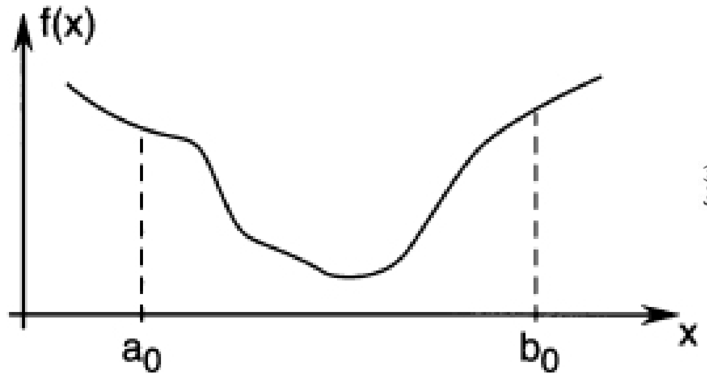
## Univariate Optimization

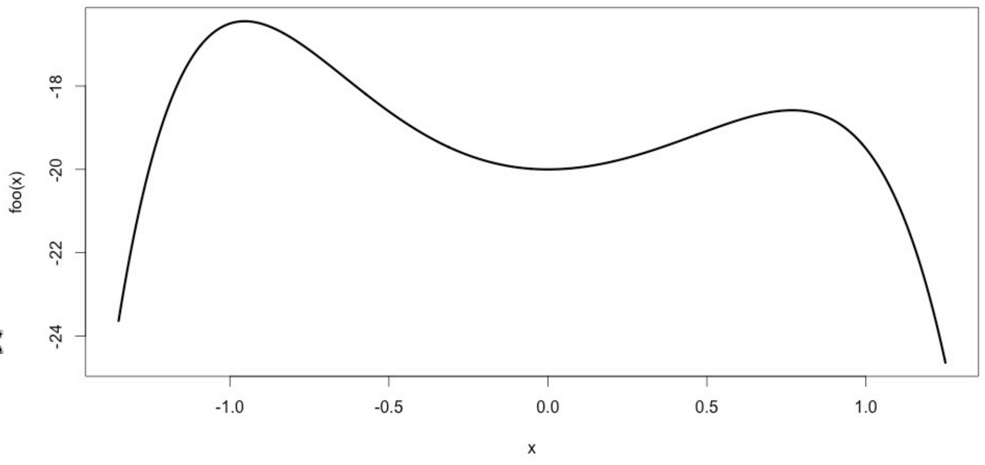
Univariate optimization refers to finding the maximum or minimum of a function $f : \Re_1 \rightarrow \Re_1$

The only property we will assume is that the function is unimodal on the domain we are interested in.

Definition:

A function *f* is *unimodal* on the interval [a, b] iff it has exactly one local minimum or maximum.





## Univariate Optimization

If it is possible (and won't take too much time) to find the max or min analytically, we should do so.

If it is not possible (or will take too much time) we can turn to iterative search algorithms.

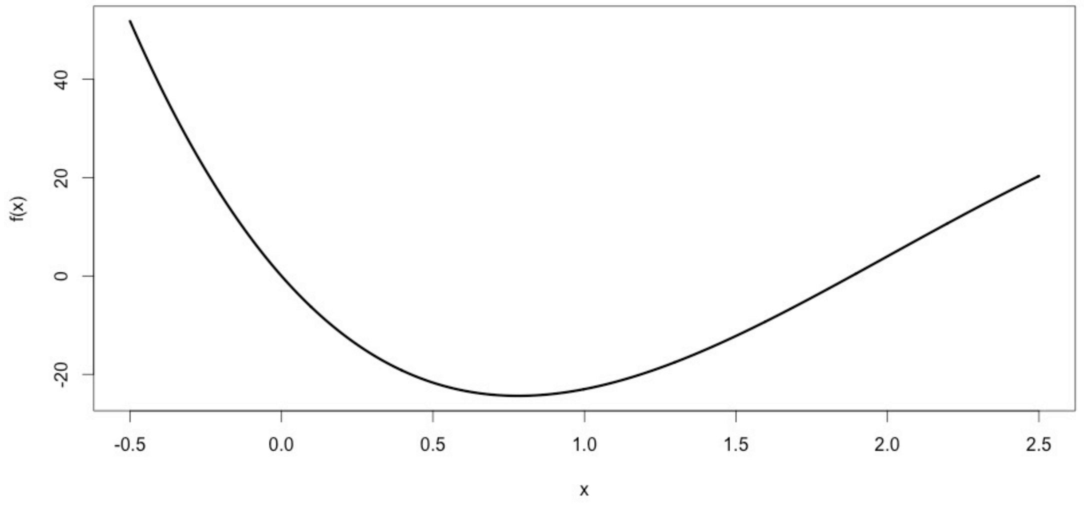
In general, we will start with some candidate value $x^{(0)}$ and generate an iterative sequence of candidate values $x^{(1)}$, $x^{(2)}$, ... where each new candidate $x^{(k+1)}$ depends on the previous value $x^{(k)}$ and the function $f$.

## Univariate Optimization

Consider this function (which we will use as an example throughout) on the domain [0, 2]:

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

```
f <- function(x)
{
  1*x^4 - 14*x^3 + 60*x^2 - 70*x
}
curve(f, from = -.5, to = 2.5, n = 1001, lwd = 3)
```



.

## Univariate Optimization

The function appears to be unimodal on [0,2]. Can we determine the minimum analytically?

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

$$f'(x) = 4x^3 - 42x^2 + 120x - 70$$

$$f'(x) = 0 = 4x^3 - 42x^2 + 120x - 70$$

Using math software we can determine the roots of the cubic equation. (Note that analytic expressions for polynomials of degree five or higher do not necessarily exist.) The roots are:

5.95719468495797

0.78088405308808 # This is the one in our domain

3.76192126195395

## Univariate Optimization

Some methods may use only the function *f*, others may use its first derivative *f′*, and others may use its second derivative *f″*.

We will look at the following univariate methods :

| **Method name** | **Uses *f, f′, f″*?** |
|---|---|
| Golden Section method | uses only *f* |
| Bisection method | uses only *f′* |
| Secant method | uses only *f′* |
| Newton's method | uses *f′* and *f″* |

## Golden Section Method

Goal: find the point M at which we have a minimum

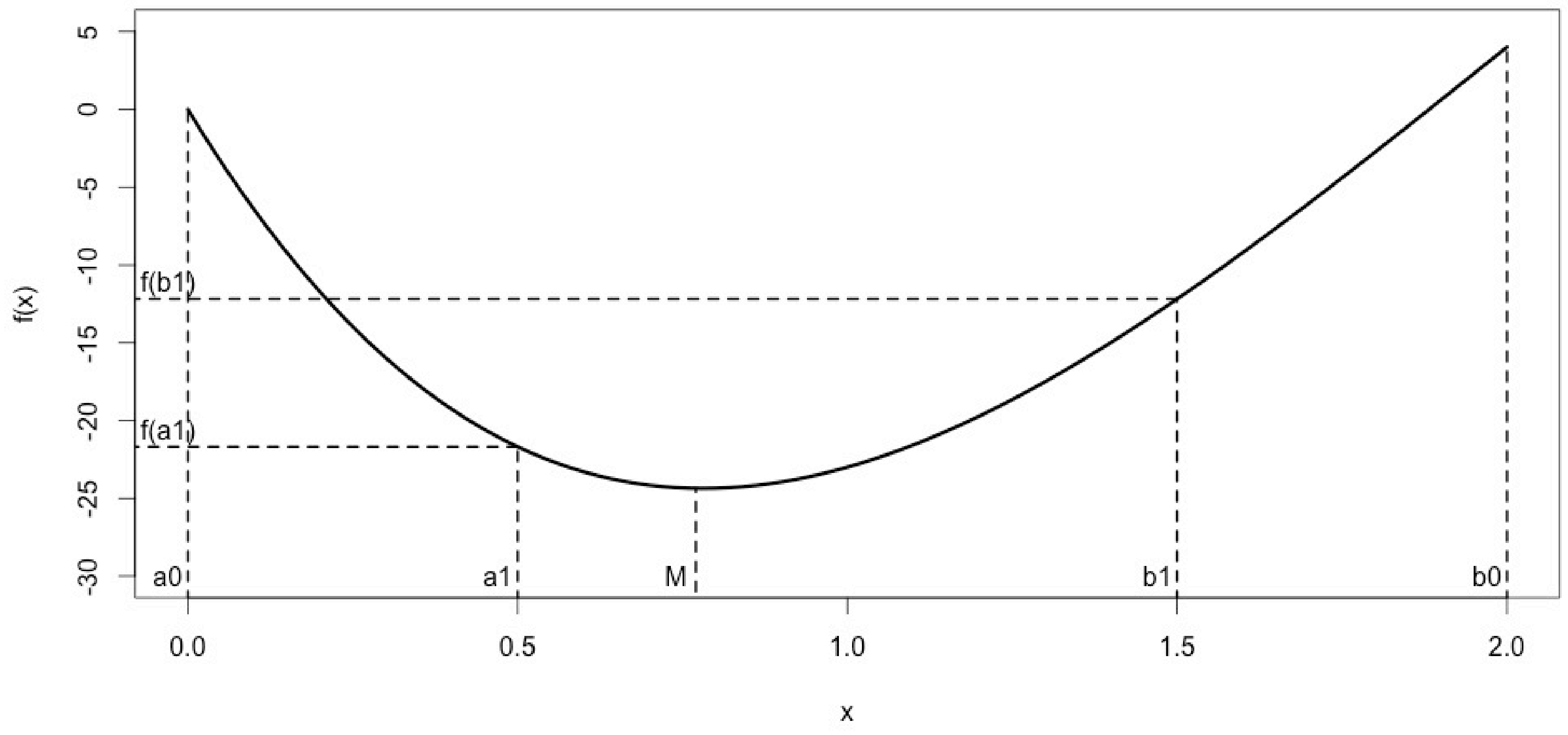
The main idea is that we start somewhere and the proceed to more narrowly close in on the answer until a 'close enough' point is reached.

In order to close in on the minimum, we need to evaluate the function at two points.

## Golden Section Method

- Start with the endpoints $a_0$ and $b_0$.
- Move in symmetrically to test points $a_1$ and $b_1$.
- Evaluate the function at the test points.
- Two possible scenarios:
  1) $f(a_1) < f(b_1)$ implies M is in $[a_0, b_1]$
  2) $f(a_1) > f(b_1)$ implies M is in $[a_1, b_0]$
- Contract the interval according to (1) or (2) above.
- Start the process again on the new interval.
- Continue until the interval size is less than the desired precision.

## Golden Section Method



## Golden Section Method

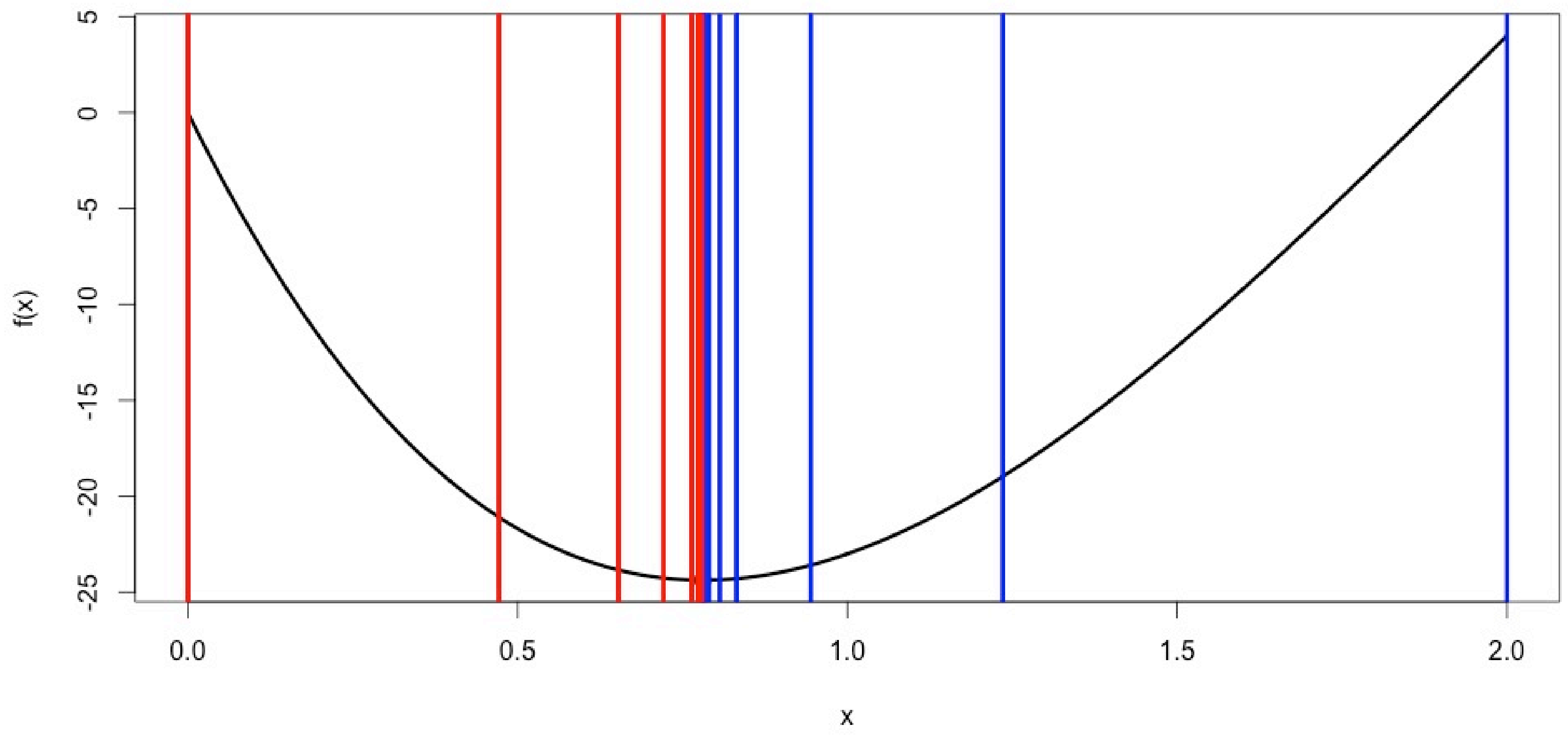
Would be nice if we only had to evaluate the function once (instead of twice) at each iteration.

This could save computational time, especially if the function we are trying to minimize is hard to evaluate.

## Golden Section Method

With $\rho = 0.382$, the length of the interval is reduced by 0.618 at each iteration. Thus, the interval may be narrowed down to size $(1 - \rho)^N = 0.618^N$ after N iterations.

## Golden Section Method



```
golden <- function(f, interval, precision = 1e-6)
{
  rho <- (3-sqrt(5))/2
  # ::: Work out first iteration here
  f_a <- f(interval[1] + rho*(diff(interval)))
  f_b <- f(interval[2] - rho*(diff(interval)))
  # ::: How many iterations to reach precision?
  N <- ceiling(log(precision/(upper - lower))/log(1-rho))
  for (i in 1:N)                                 # index the number of iterations
  {
    if (f_a < f_b)
    {
      # ::: FILL IN CODE HERE
    } else{
      if (f_a >= f_b)
      {
        # ::: FILL IN CODE HERE
      } }
  }
  interval
}
```

## Bisection Method

As before, we again assume $f$ is unimodal on the domain of interest. We also assume that $f$ is continuously differentiable on the domain.

The derivative of $f$ at a point $x_0$ tells us the slope of the function at that point.

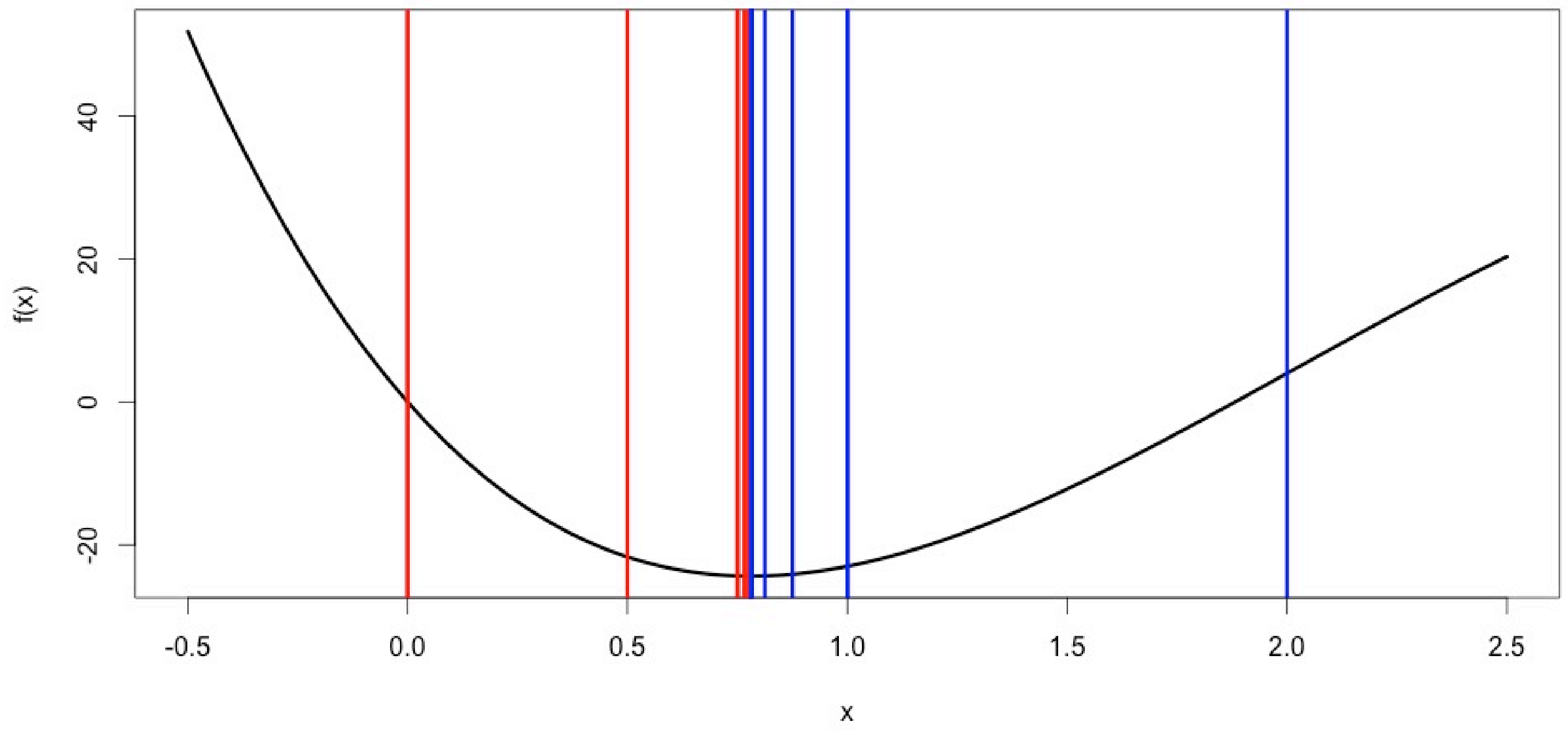
Thus, if we pick a point in the domain and find that the slope of $f$ is negative there, we know that the function is *decreasing* so we look to the right for the minimizer.

Of course, if $f'(x_0) = 0$, we have found the minimizer.

## Bisection Method

Faster convergence: the bisection method reduces the size of the uncertainty interval by a factor of 0.5 (as opposed to 0.328 in the golden section method).

## Bisection Method



## Bisection Method

```
bisection <- function(f_prime, int, precision = 1e-7)
{
  N <- ceiling(log(precision/(diff(int)))/log(.5))
  f_prime_a <- f_prime(int[1] + diff(int)/2)
  for (i in 1:N)
  {
    if(f_prime_a < 0)
    {
      # ::: FILL IN CODE HERE
    } else
      if(f_prime_a > 0)
      {
        # ::: FILL IN CODE HERE
      } else
        if(f_prime_a == 0)
        {
          # ::: FILL IN CODE HERE
        }
    # ::: FILL IN CODE HERE (UPDATE)
  }
  int
}
```

## Newton's Method

Same assumptions as the bisection method, although we additionally assume that the derivative is continuously differentiable. That is, both $f'$ and $f''$ are available on the domain.

The main idea behind Newton's method is that at each point, since the second derivative is available, we can fit a quadratic function, $q$, so that the first and second derivatives of $f$ and $q$ are the same.

Then, we minimize $q$ and move the testpoint to its minimizer.

## Newton's Method

$$\text{At } x_k, \ \text{let } q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2$$

Check:

Does $q(x)$ go through the point $(x_k, f(x_k))$?

Is the first derivative of $q(x) = f'(x)$ when $x = x_k$?

Is the second derivative of $q(x) = f''(x)$ when $x = x_k$?

# Newton's Method

To minimize *q*, set its derivative equal to zero:

$$0 = q'(x) = f'(x_k) + f''(x_k)(x - x_k)$$

$$x = x_k - \frac{f'(x_k)}{f''(x_k)}$$

We set the next test point equal to the expression above. Note we only use *f'* and *f''*, not *f* itself.

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

## Newton's Method

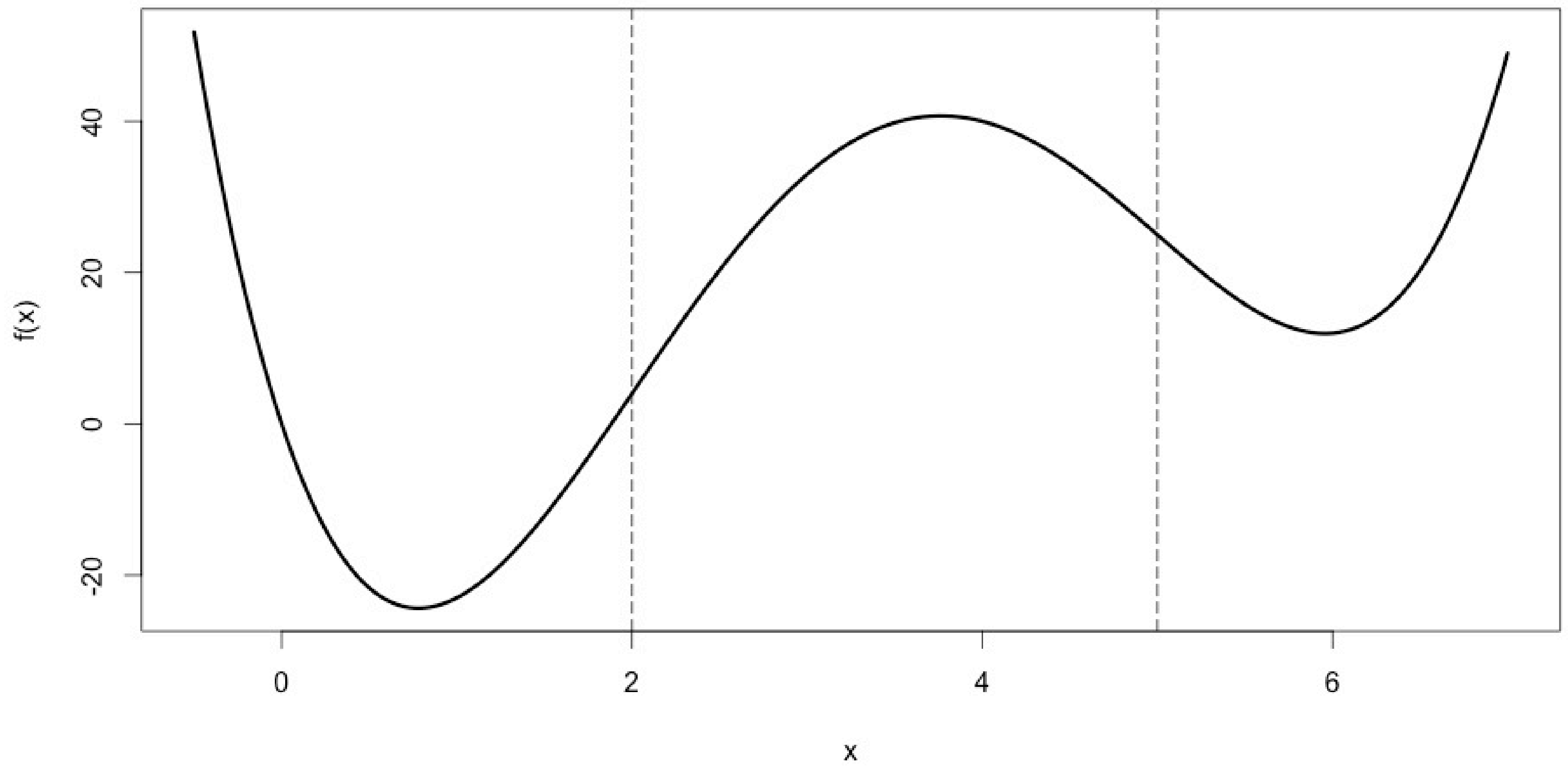
Further differences between Newton's method and the golden section and bisection methods:

- Need to specify a starting point for Newton's method. Should be an educated guess. The selection of a very poor starting point can lead to nonconvergence.
- No clean expression for the number of iterations. Must specify stopping point of the form:

```
    if(abs(x_new-x_old) < epsilon) stop
```

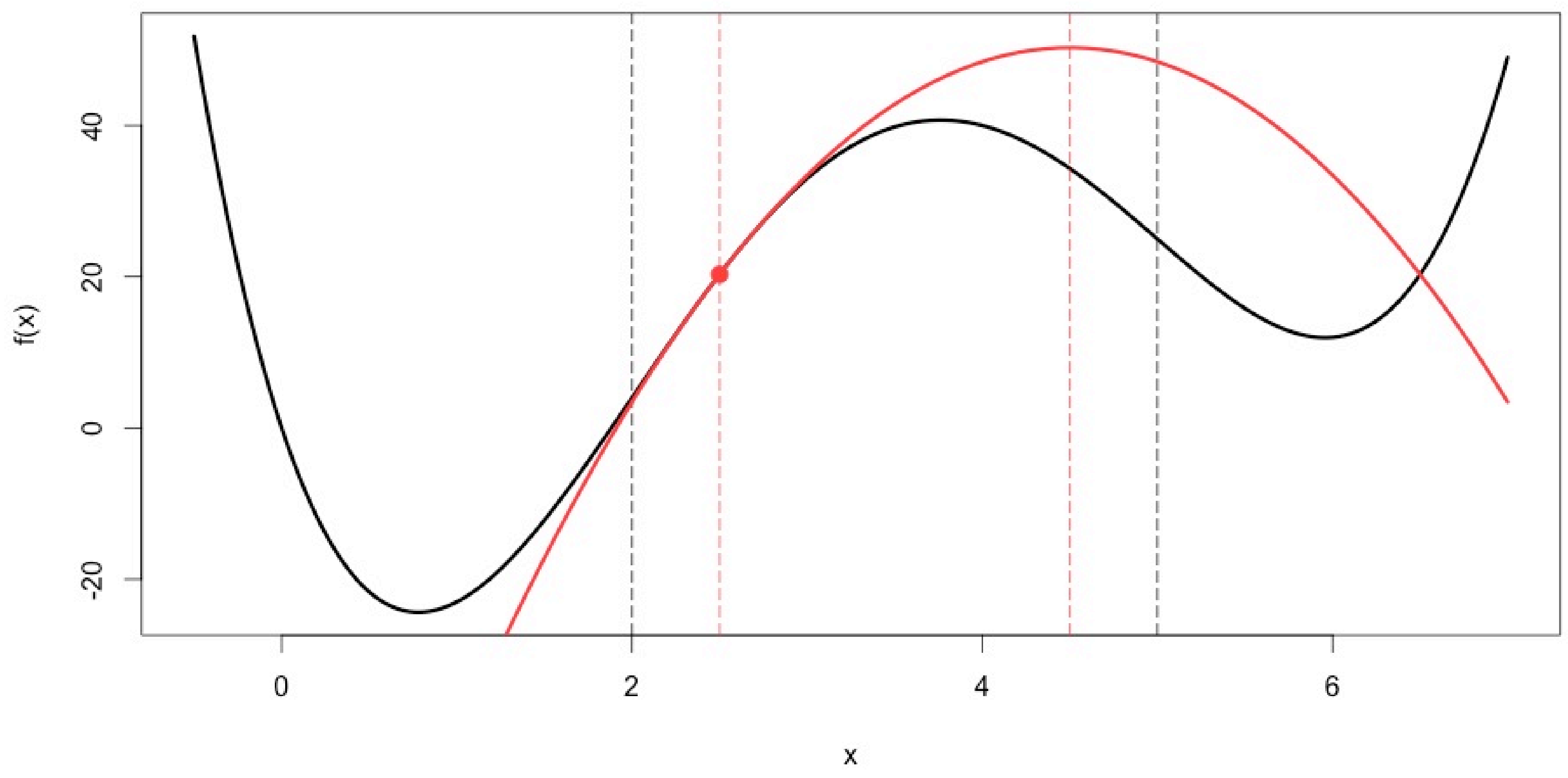
## Newton's Method

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x, \quad \text{where } x \in [-0.5, 7]$$



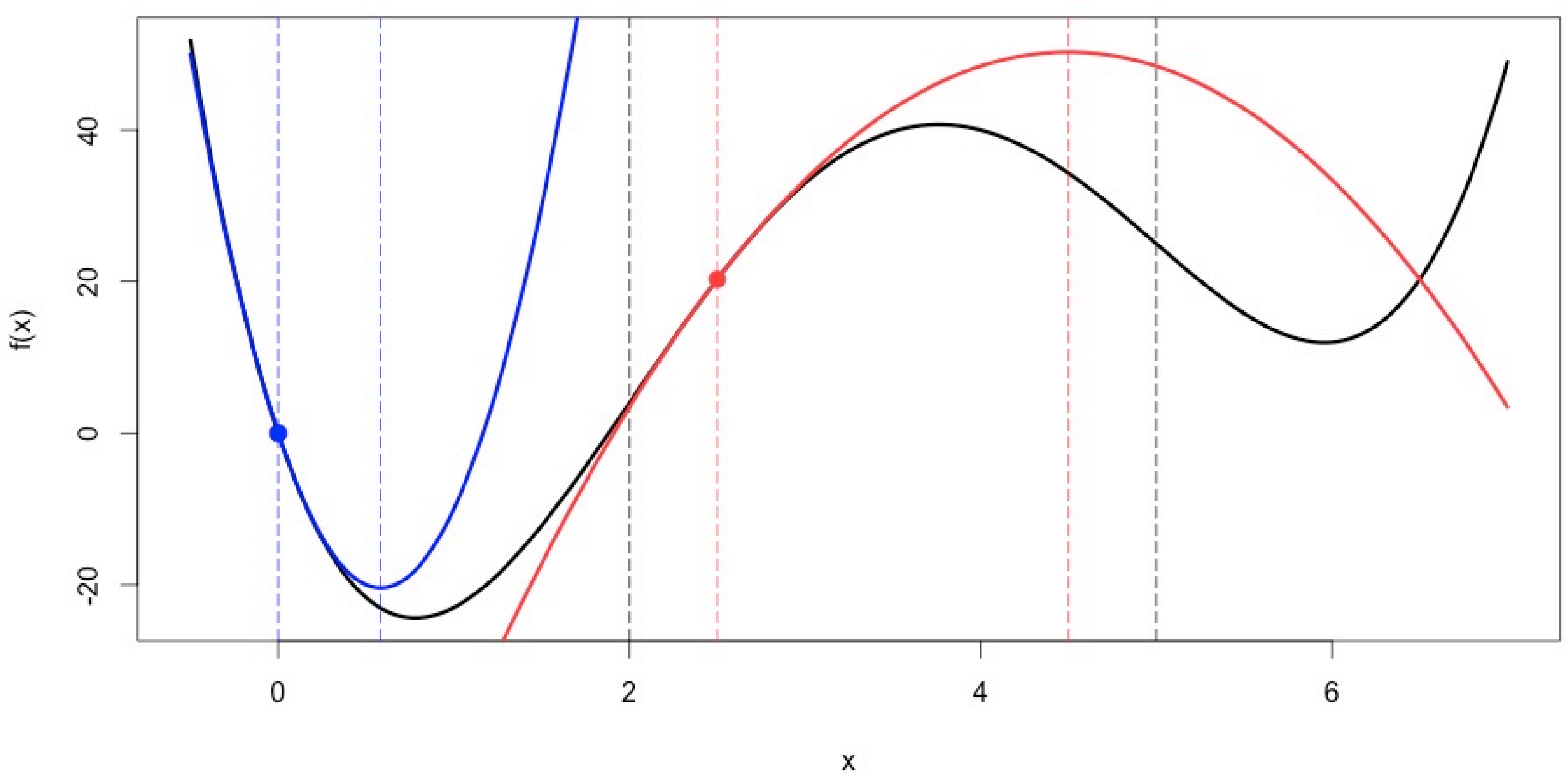
# Newton's Method

$q(x) = f(2.5) + f'(2.5)(x-2.5) + \frac{1}{2} f''(2.5)(x-2.5)^2$ has max at 4.5.



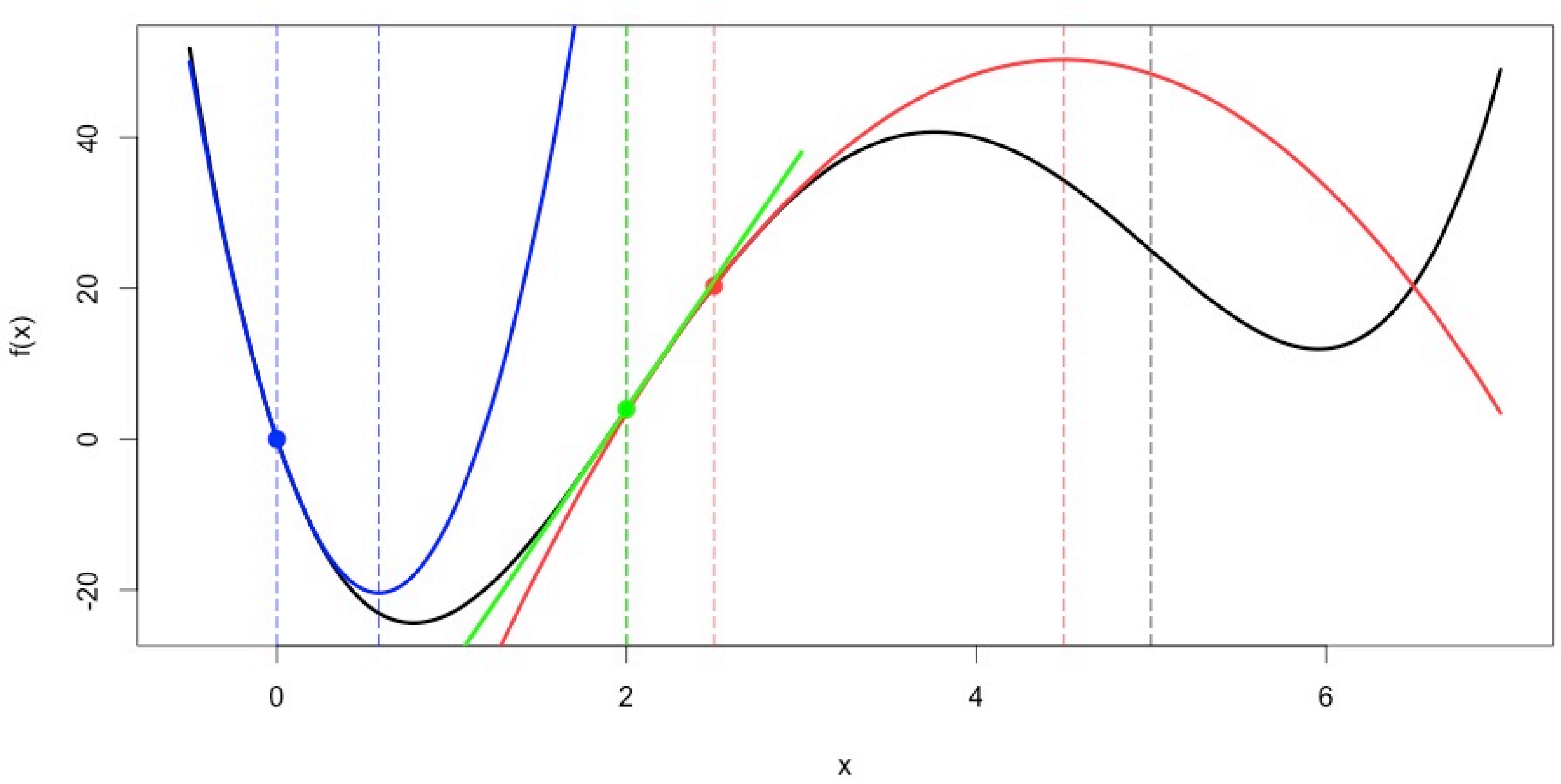
## Newton's Method

$$q(x) = f(0) + f'(0)(x-0) + \frac{1}{2}f''(0)(x-0)^2$$ has a min at 0.5833.



# Newton's Method

What happens at an inflection point (i.e., where $f'' = 0$)?



## Newton's Method

Newton's method finds the solution to the equation *f'*(*x*) = 0. If we want it to find the solution to the equation *f*(*x*) = 0 instead, we can simply make a substitution.

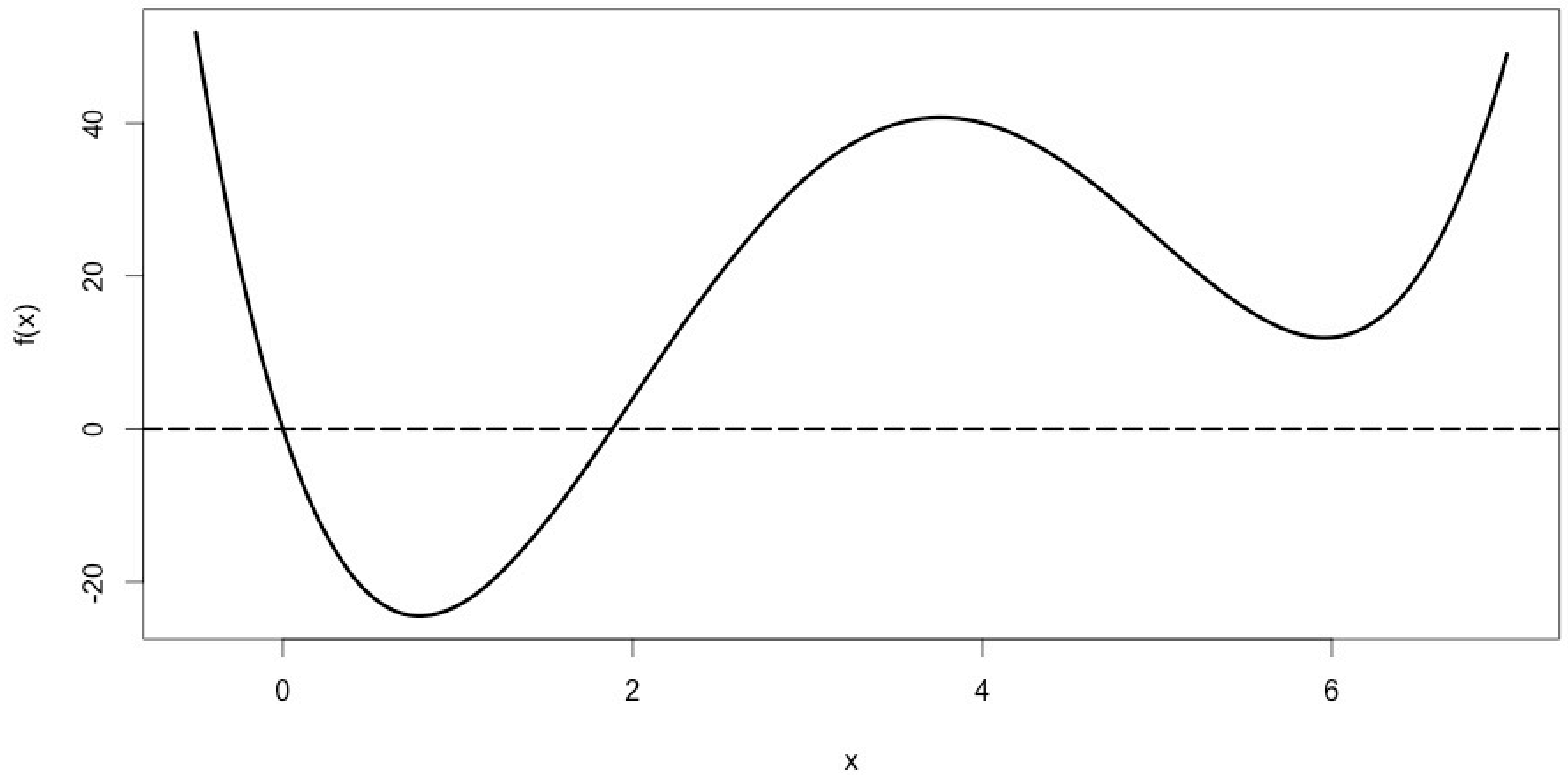
Called *Newton's method of tangents* when used this way.

That is, Newton's method can also be used to find the zeros of a function by using *f* instead of *f'* and *f'* instead of *f''* in the formula.

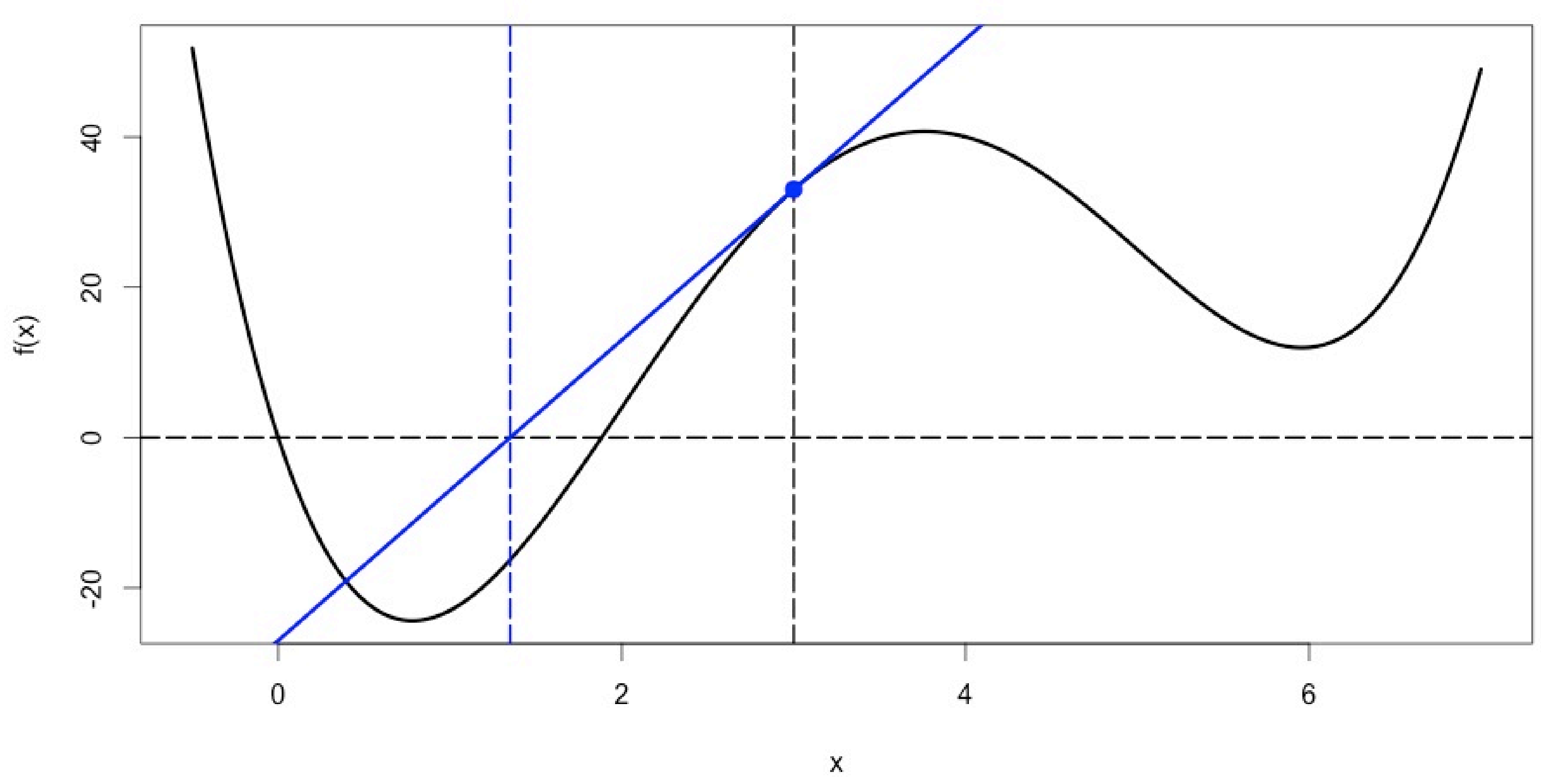
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

## Newton's Method

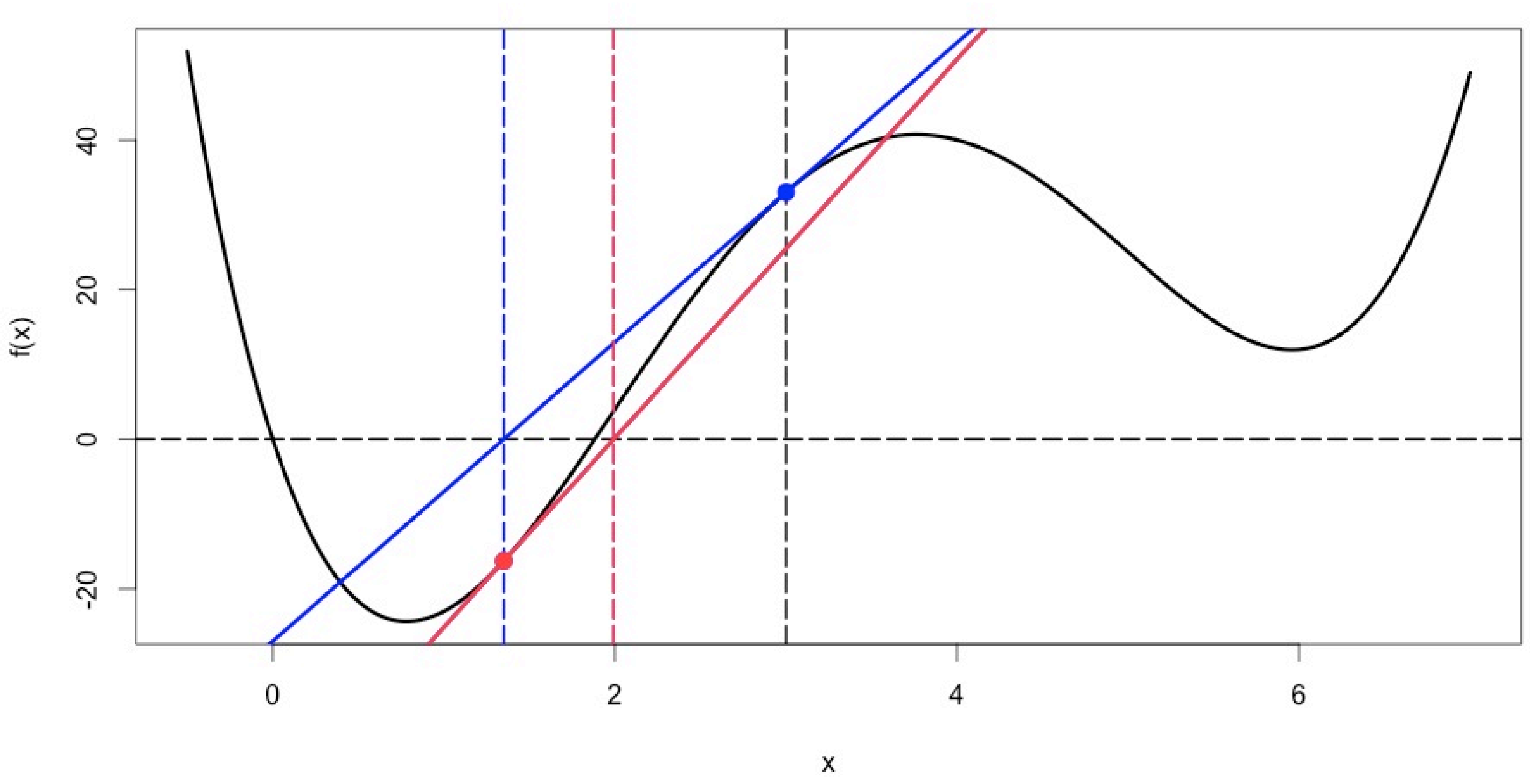
$$f(x) = x^4 - 14x^3 + 60x^2 - 70x, \quad \text{where } x \in [-0.5, 7]$$



# Newton's Method



# Newton's Method



## Newton's Method

```
newton <- function(f_prime, f_dbl, precision = 1e-6,
                      start)
{
  # ::: f_prime is first derivative function
  # ::: f_dbl is sec der function
  # ::: start is starting 'guess'

  i <- 1 # ::: use 'i' to print iteration number
  while (abs('something here') > precision)
  {
    # ::: redefine variables and calculate new estimate

    # ::: keep track of iteration history
    print(paste0("Iteration ", i, "; Estimate = ", x_new) )
    i <- i + 1
  }
  output
}
```

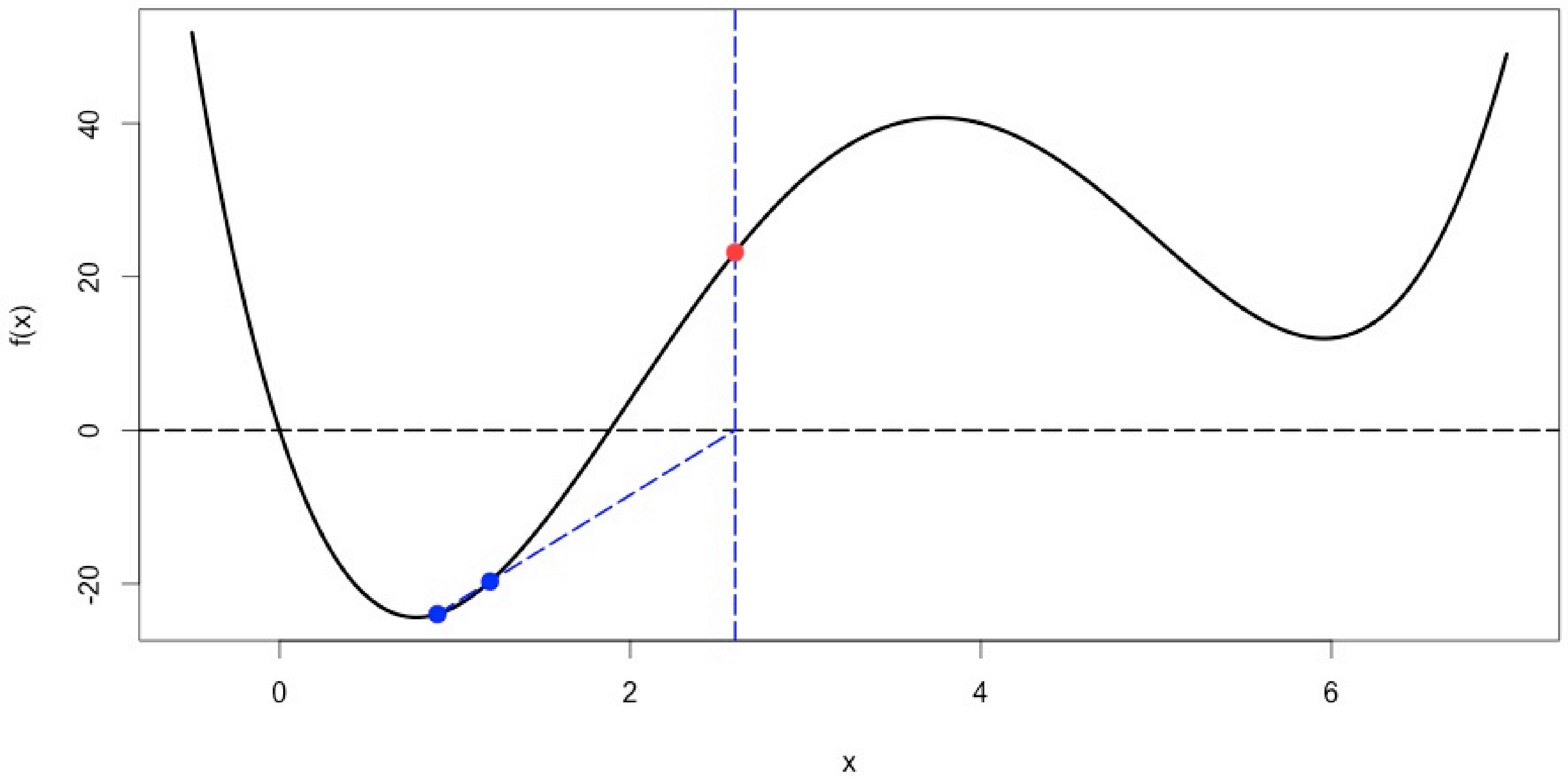
## Secant Method

One of the weaknesses of Newton's method for root finding is that it requires the first and second derivatives.

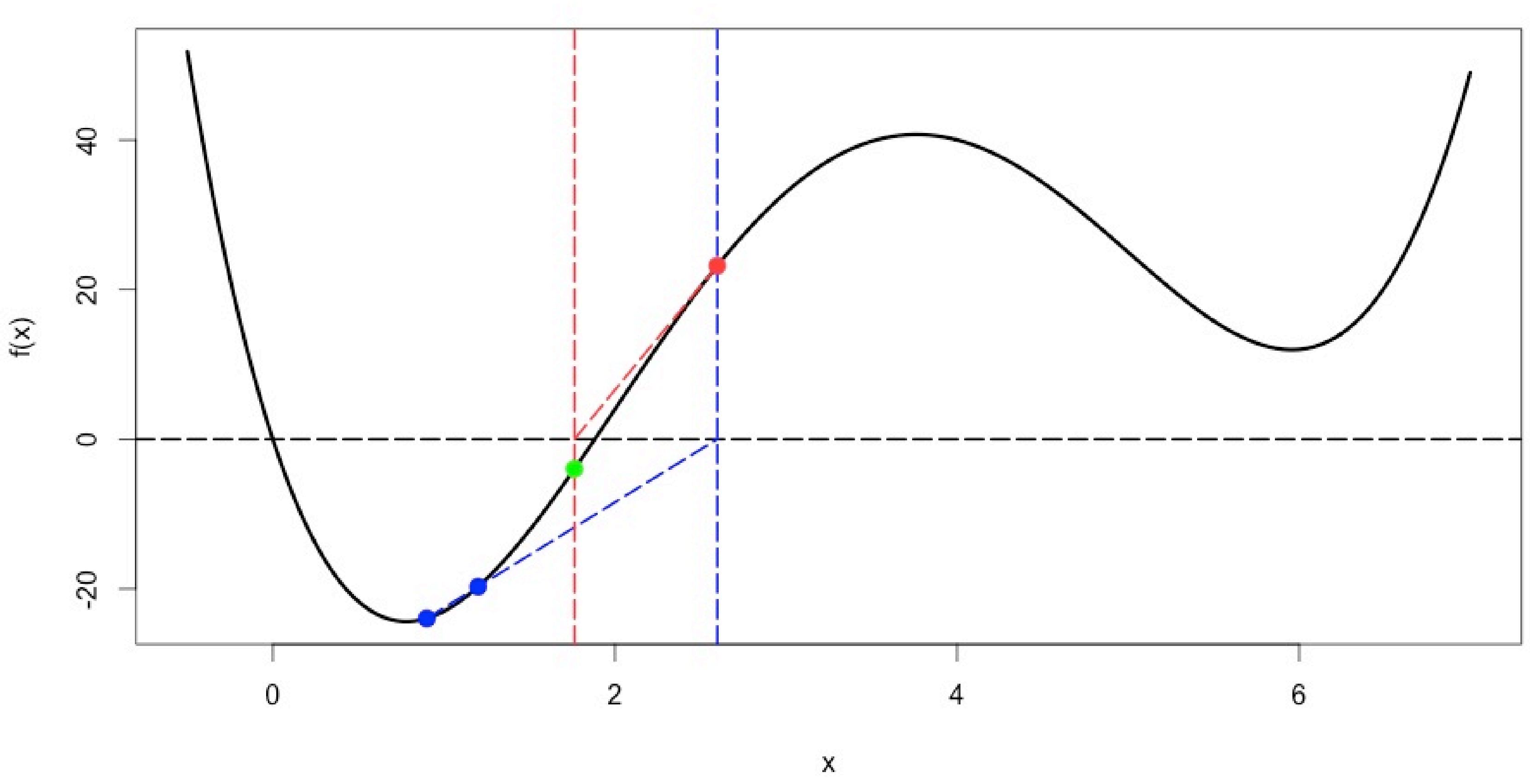
The *secant* method is an attempt to retain the speed advantage of Newton's method, while not requiring the use of the second derivative.

The value of the second derivative at $x_k$ represents the slope of the first derivative at $x_k$. Therefore, it can be approximated if we use another point on the curve.

## Secant Method



# Secant Method



## Secant Method

For root finding, the secant method only uses *f*. For root finding, the secant method update rule is

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k)$$

WARNING: with both Newton's method and the secant method, 'good' starting approximations are necessary. Otherwise results can be way off or fail to converge completely.

This can sometimes be detected by examining iteration history.