Comp 311
Functional Programming

Nick Vrvilo, Two Sigma Investments
Robert “Corky” Cartwright, Rice University

October 24, 2017

Announcements

Halite-II officially launched:
https://halite.io/

We'll give extra credit for students who write a decent bot using Scala (not limited to Core Scala)

More details and Hackathon info next time

Generative Recursion

Generative vs Structural Recursion

- The functions we have studied to this point have (mostly) followed a common pattern:
  - Break into cases
  - Decompose data into components
  - Process components (usually recursively)
- Functions that follow this pattern are referred to as *structurally recursive functions*

Generative vs Structural Recursion

- Some problems are not amenable to solution by recursive descent
  - Instead, a deeper insight or "eureka" is required
  - Often a result from mathematics or computer science must be applied to discover important structure
  - Consider Euclid's Algorithm for GCD
- The discovery of these insights and construction of solutions using them is the study of *algorithms*

Generative vs Structural Recursion

- Typically the design of an algorithm distinguishes two kinds of problems:
  - Base cases (or trivially solvable cases)
  - Problems that can be reduced to other problems of the same form
- The design of algorithms using this approach is referred to as *generative recursion*

Square Roots

- We would like to define a function `sqrt` that takes a non-negative value of type `Double` and returns the square root of that value

$$x^2 = 2$$

- There is no obvious way to apply structural recursion to this problem

Square Roots

- We would like to define a function `sqrt` that takes a non-negative value of type `Double` and returns the square root of that value

$$x^2 - 2 = 0$$

- There is no obvious way to apply structural recursion to this problem

Newton's Method

- We can use derivatives to find successively better approximations to the zeroes of a real-valued function:

$$f(x) = 0$$

Newton's Method

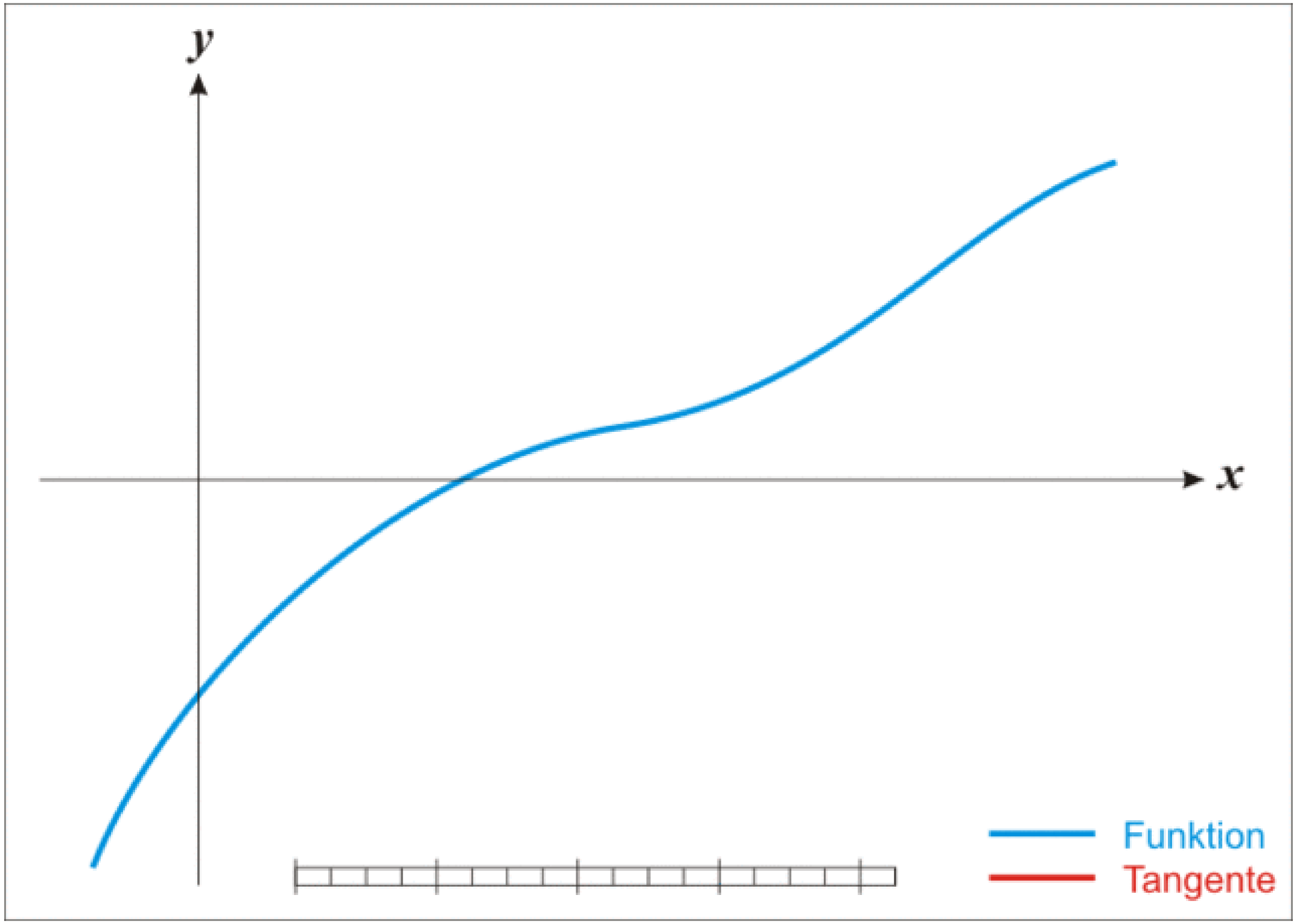
- We start with some guess for a value of x

$$x_0 = \texttt{guess}$$

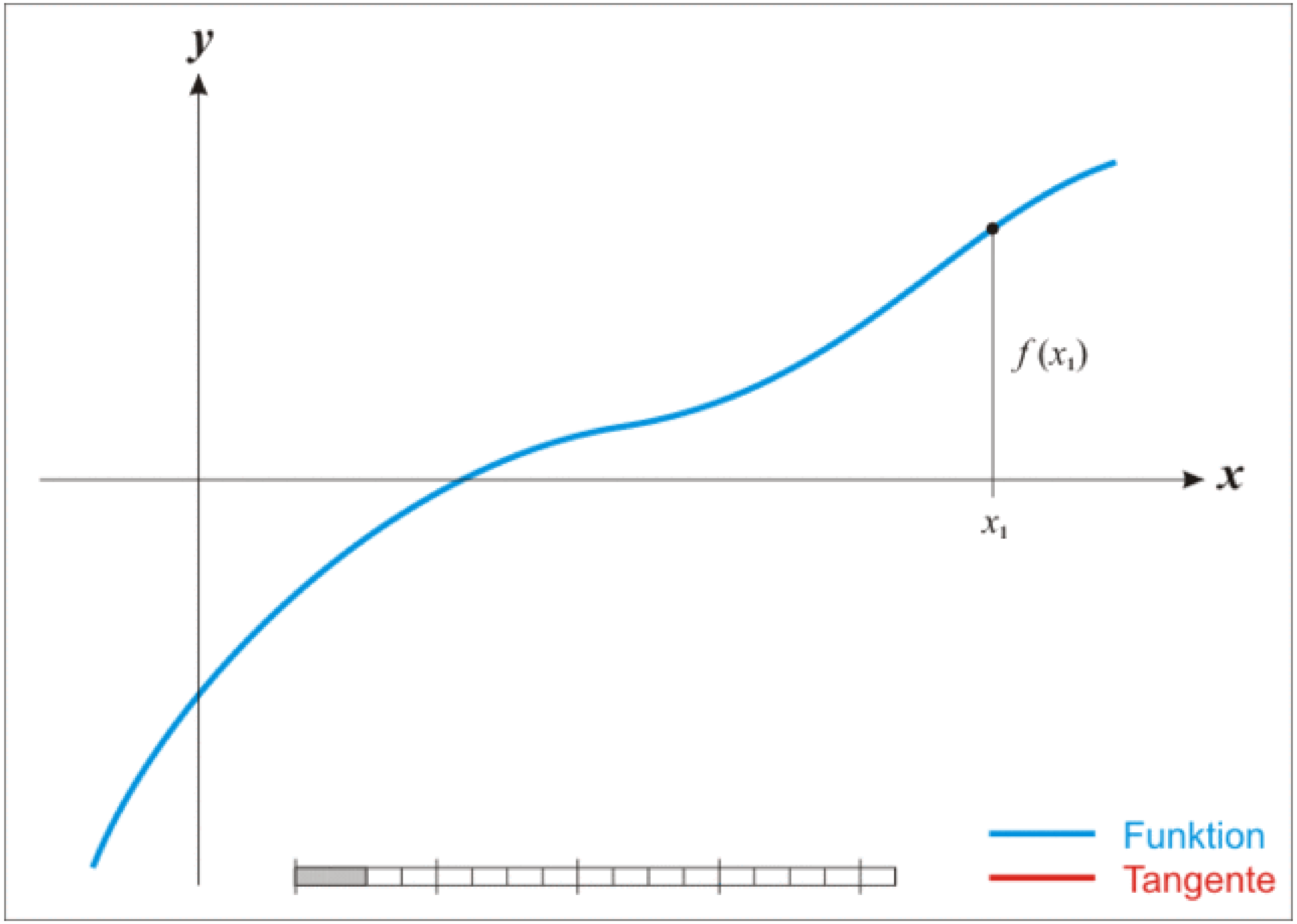
Newton's Method

- Then we construct a better approximation with the following formula:

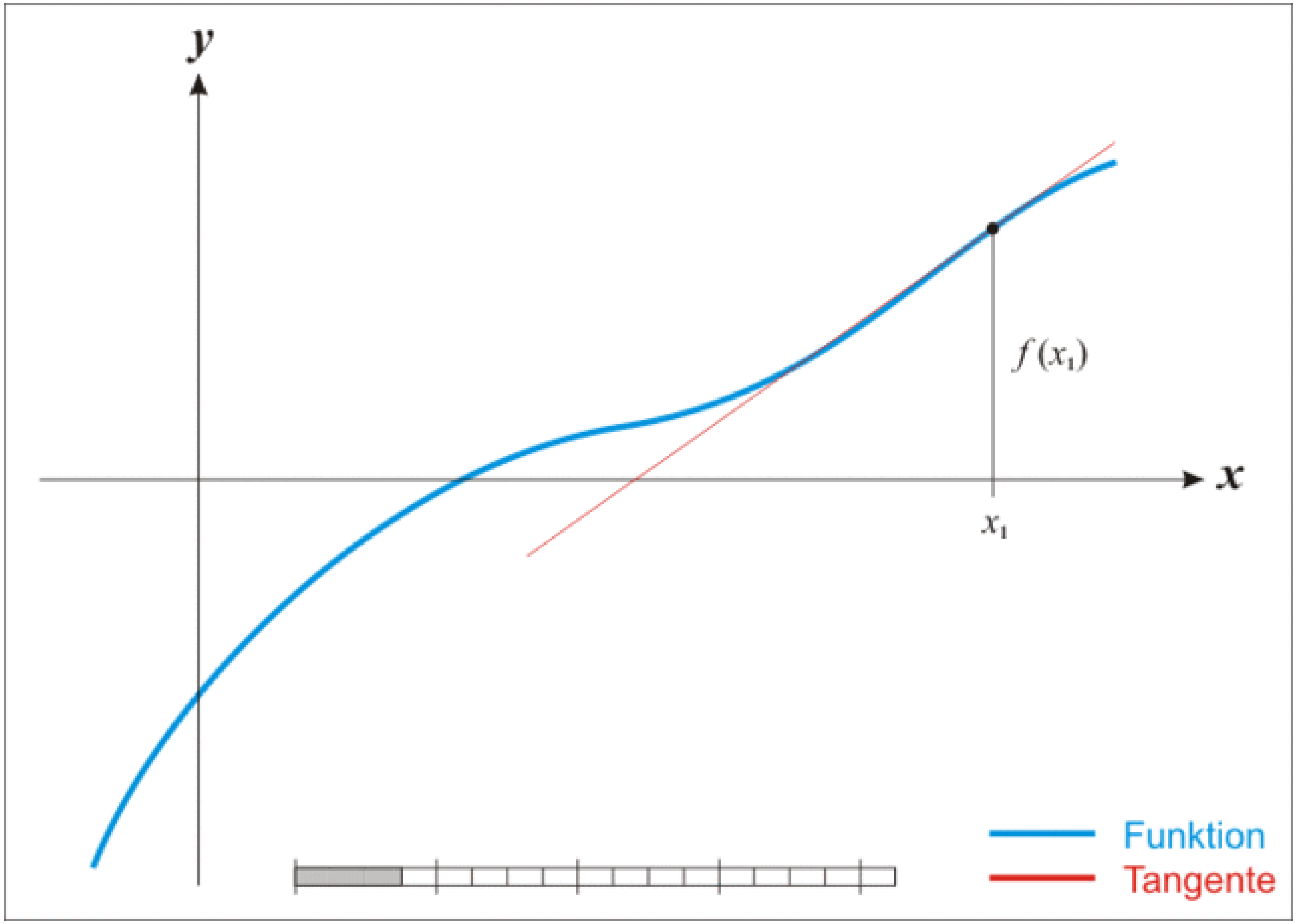
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



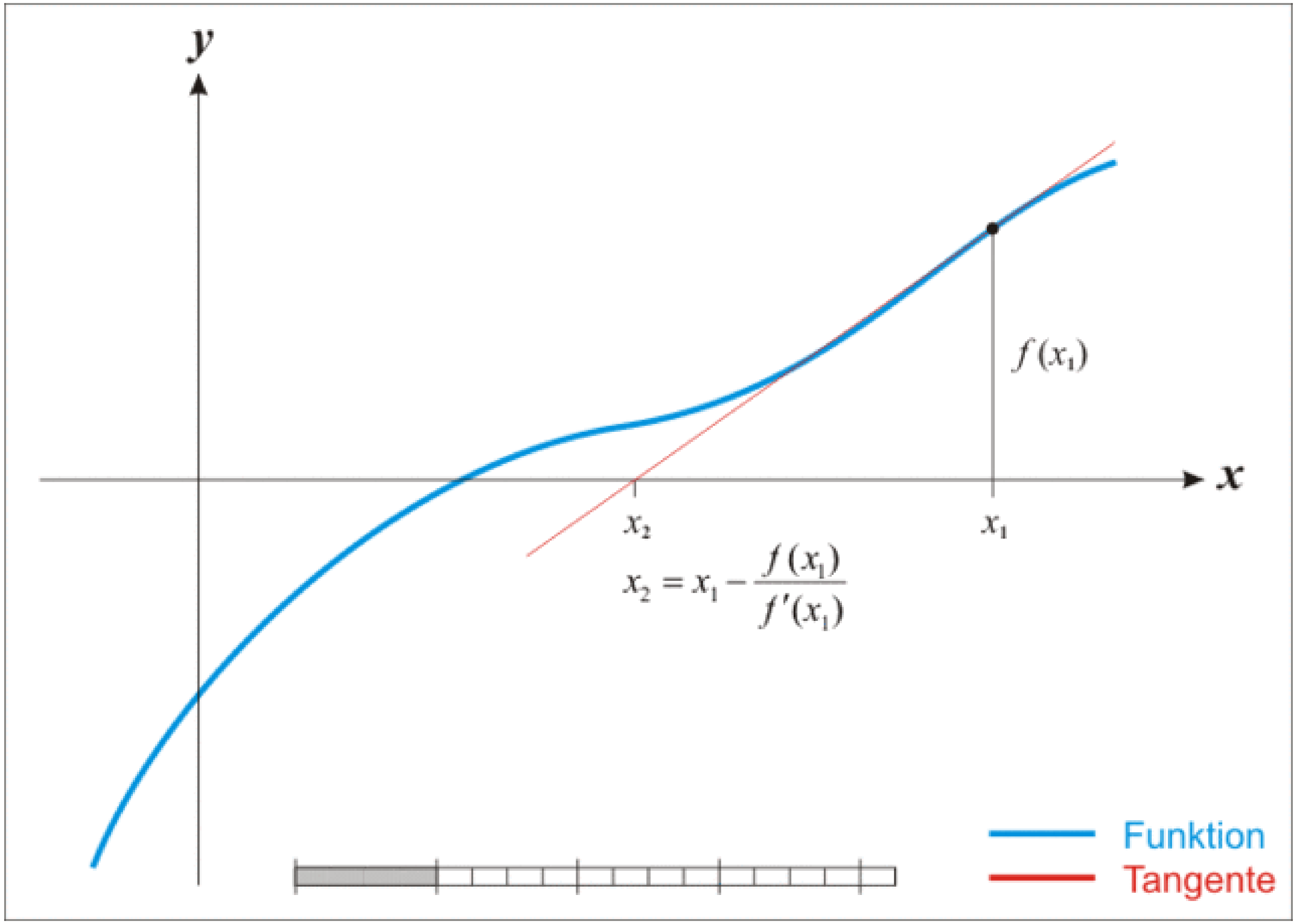
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



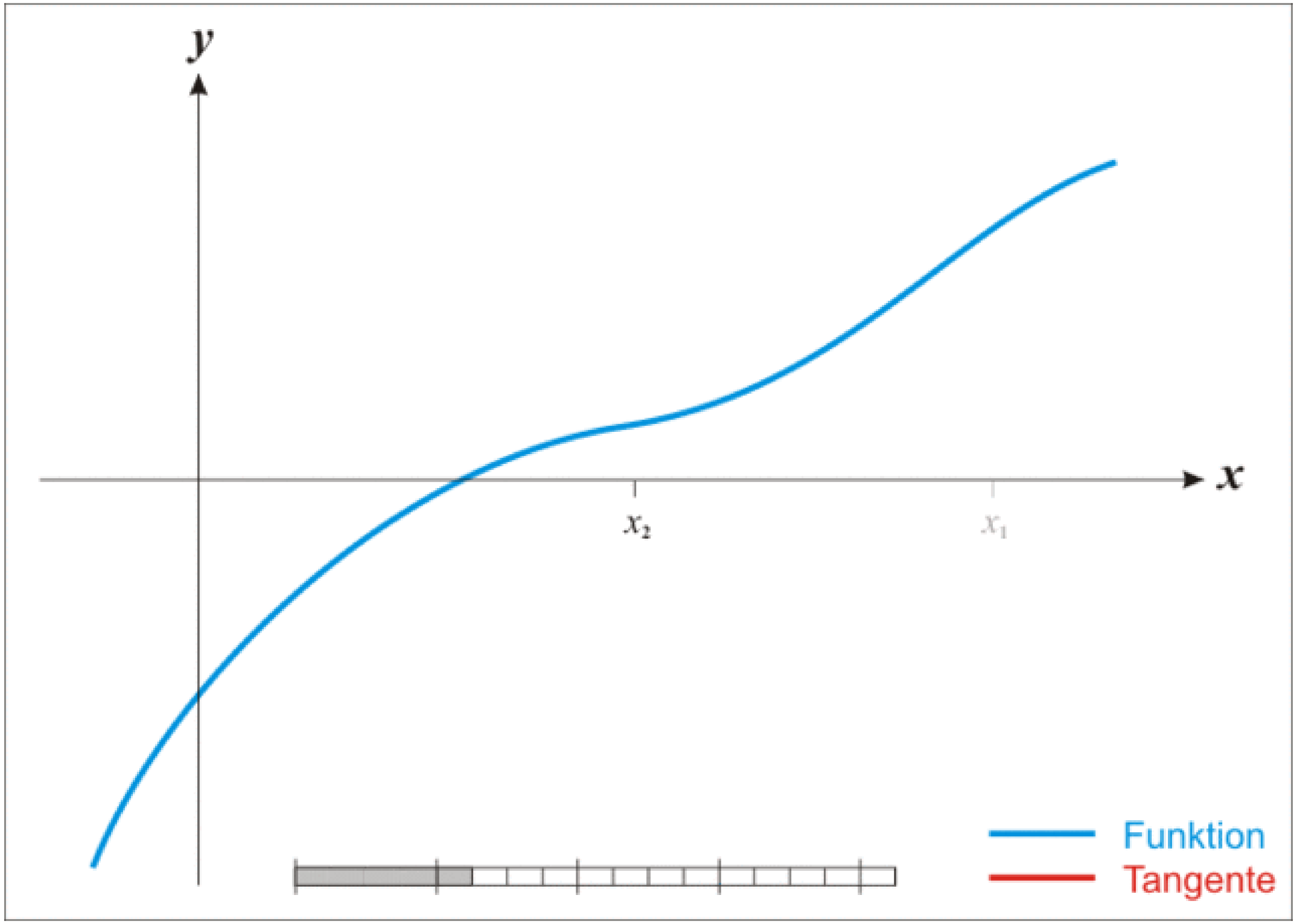
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



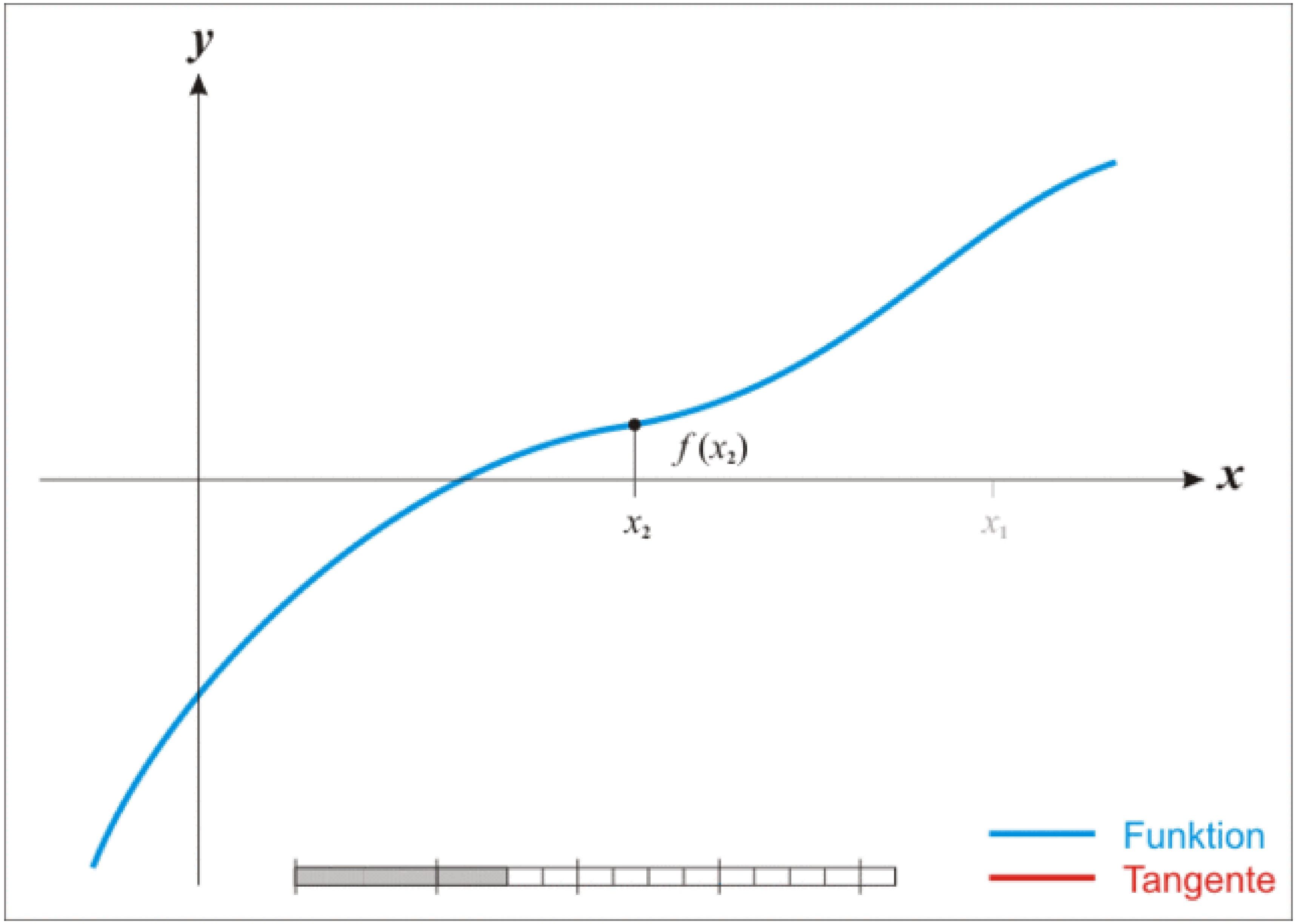
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



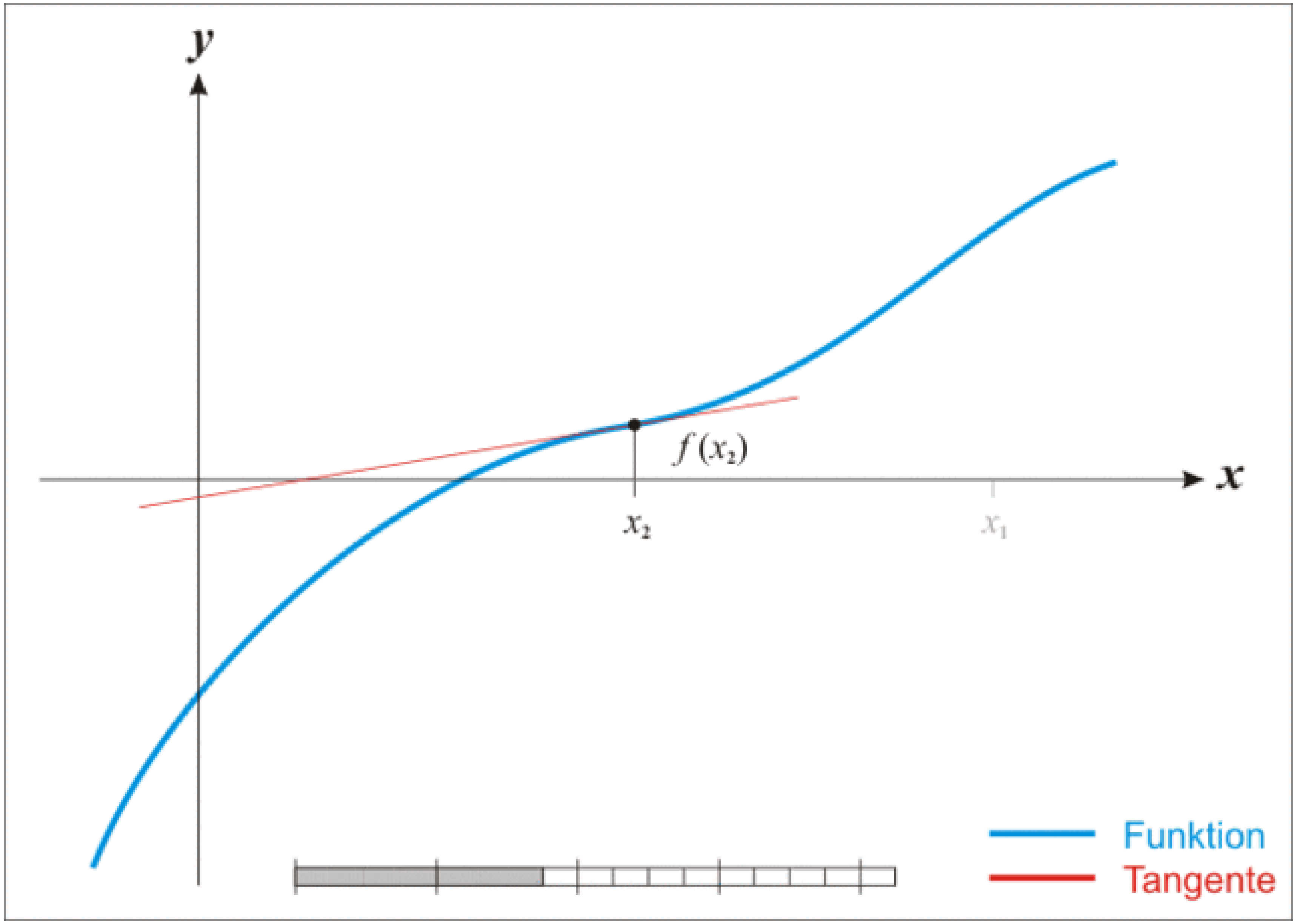
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



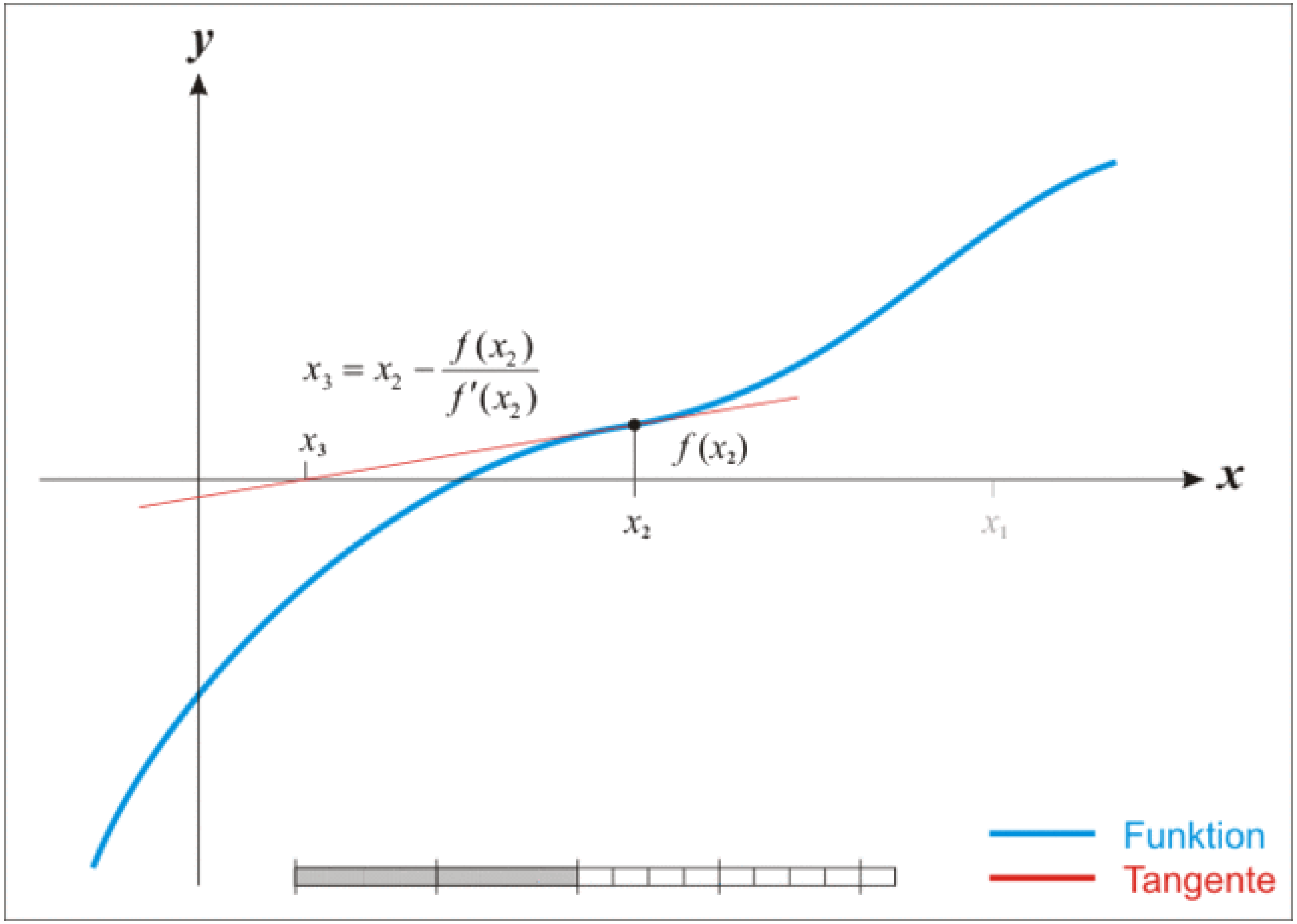
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



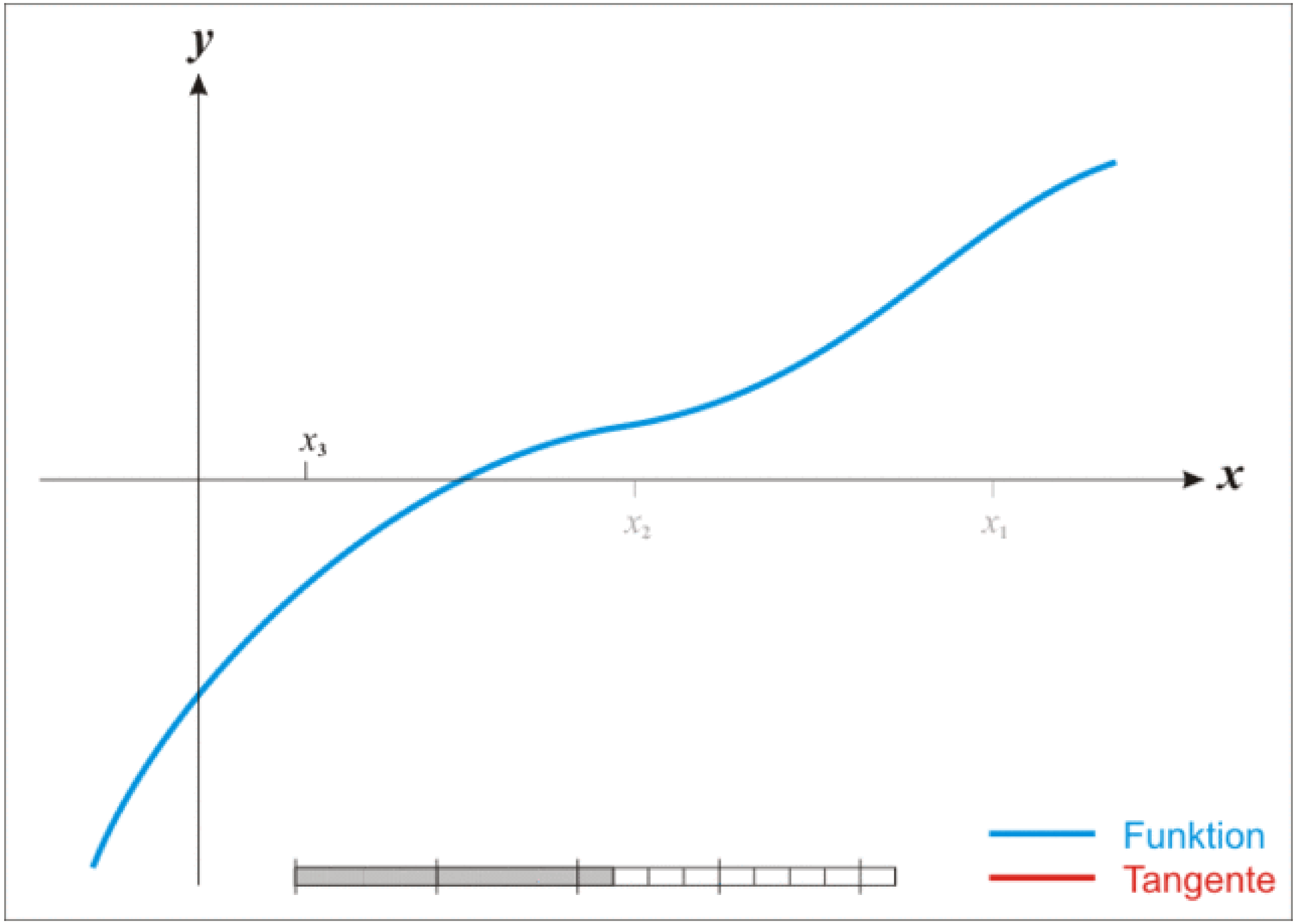
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



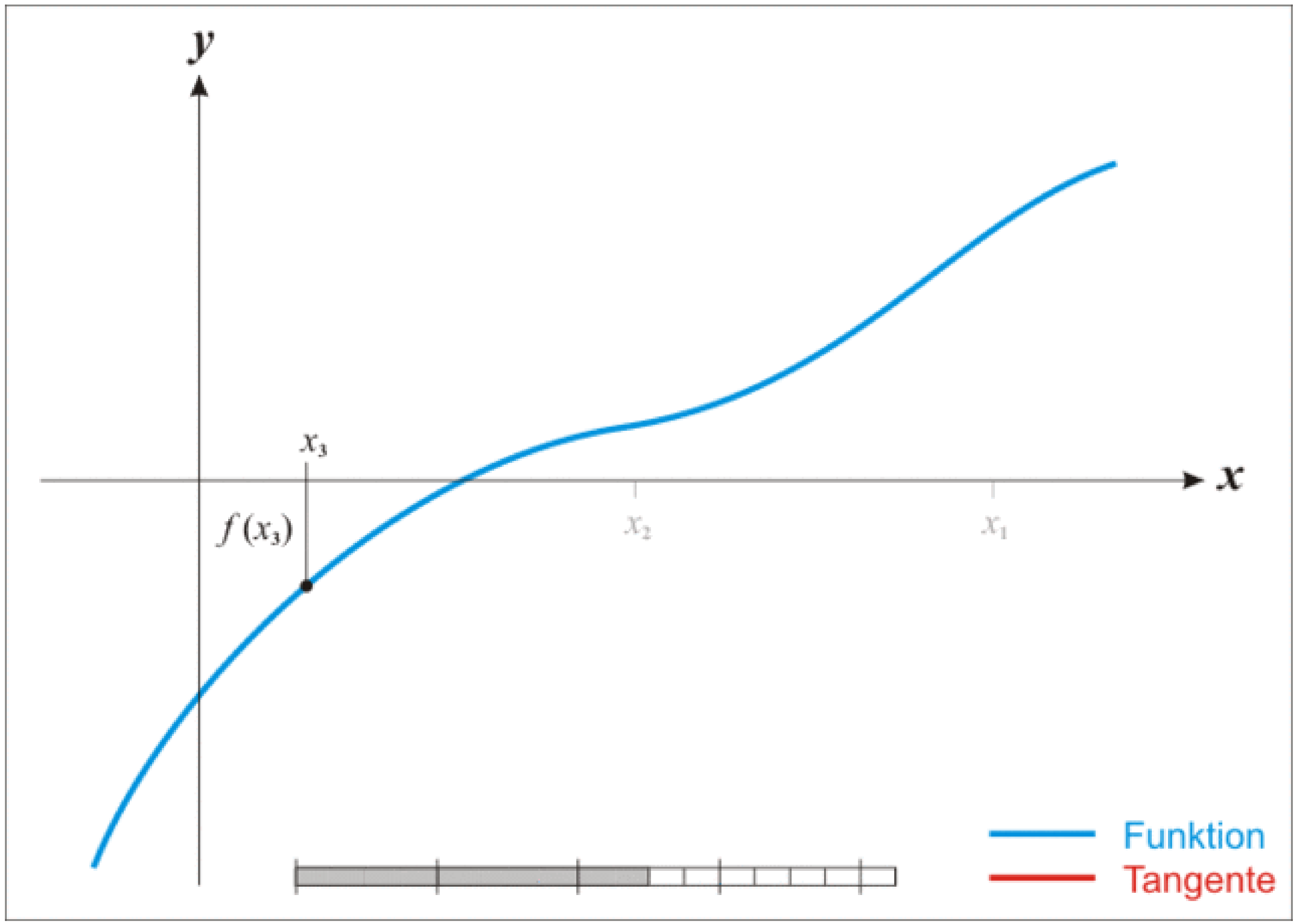
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



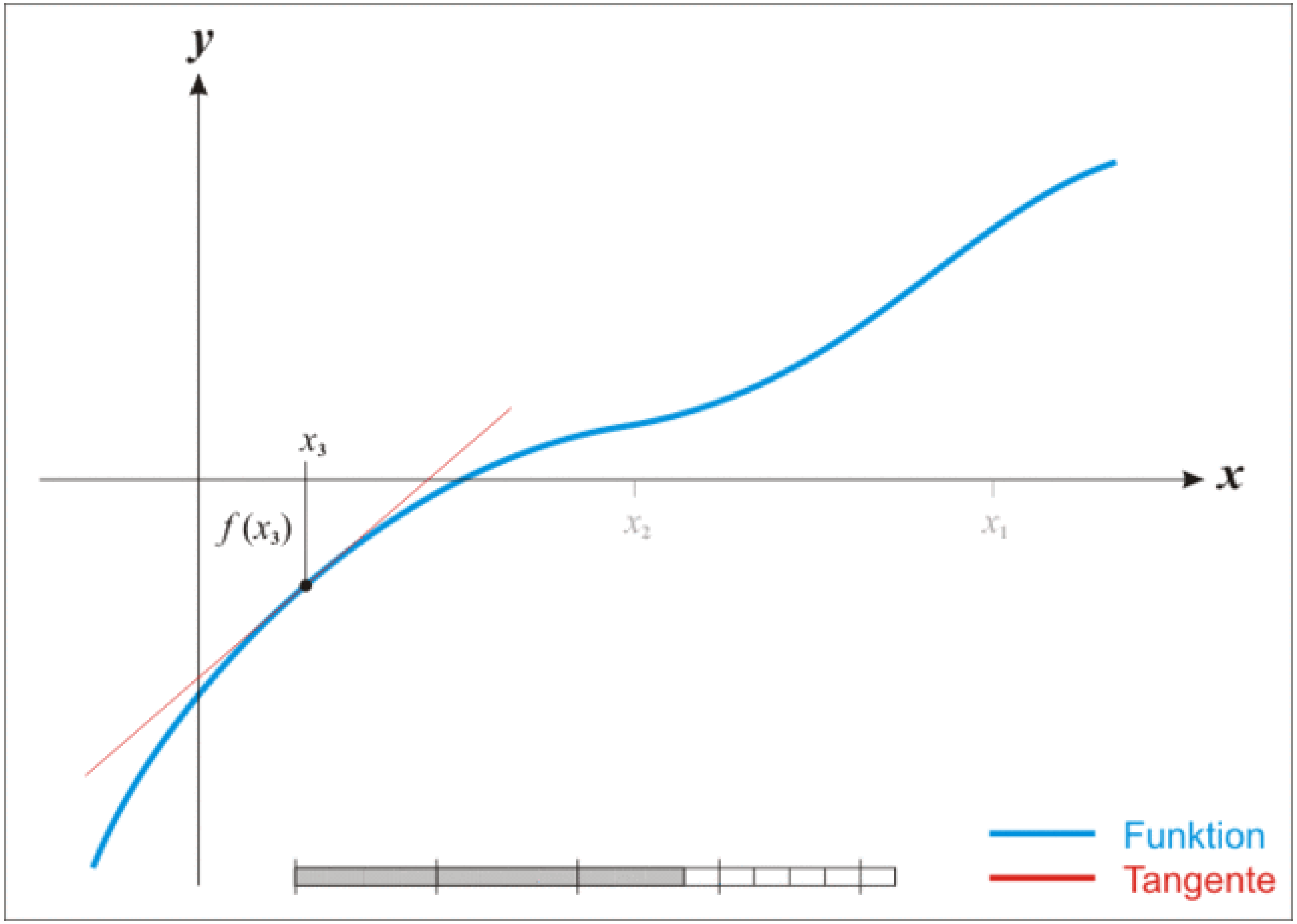
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



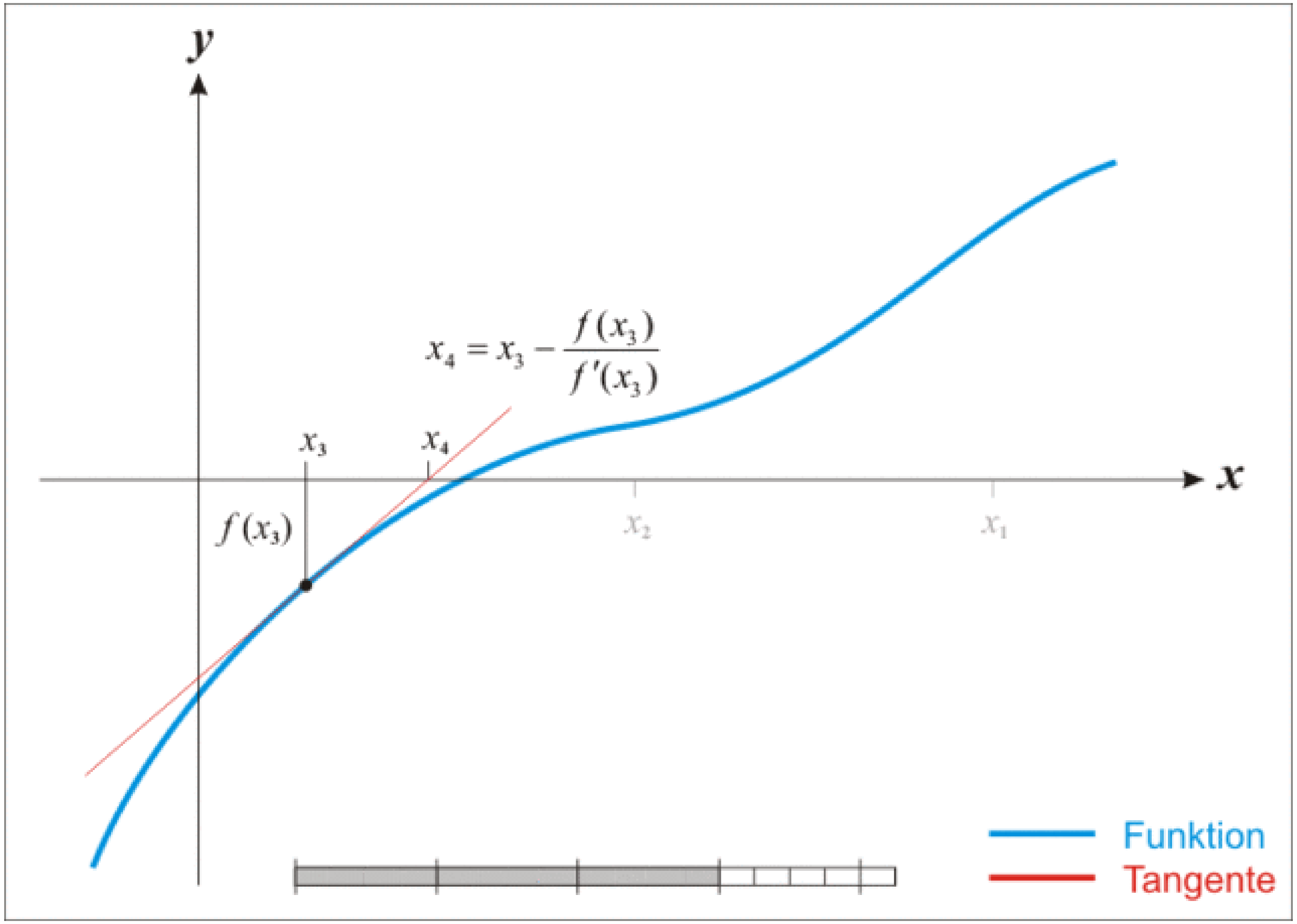
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



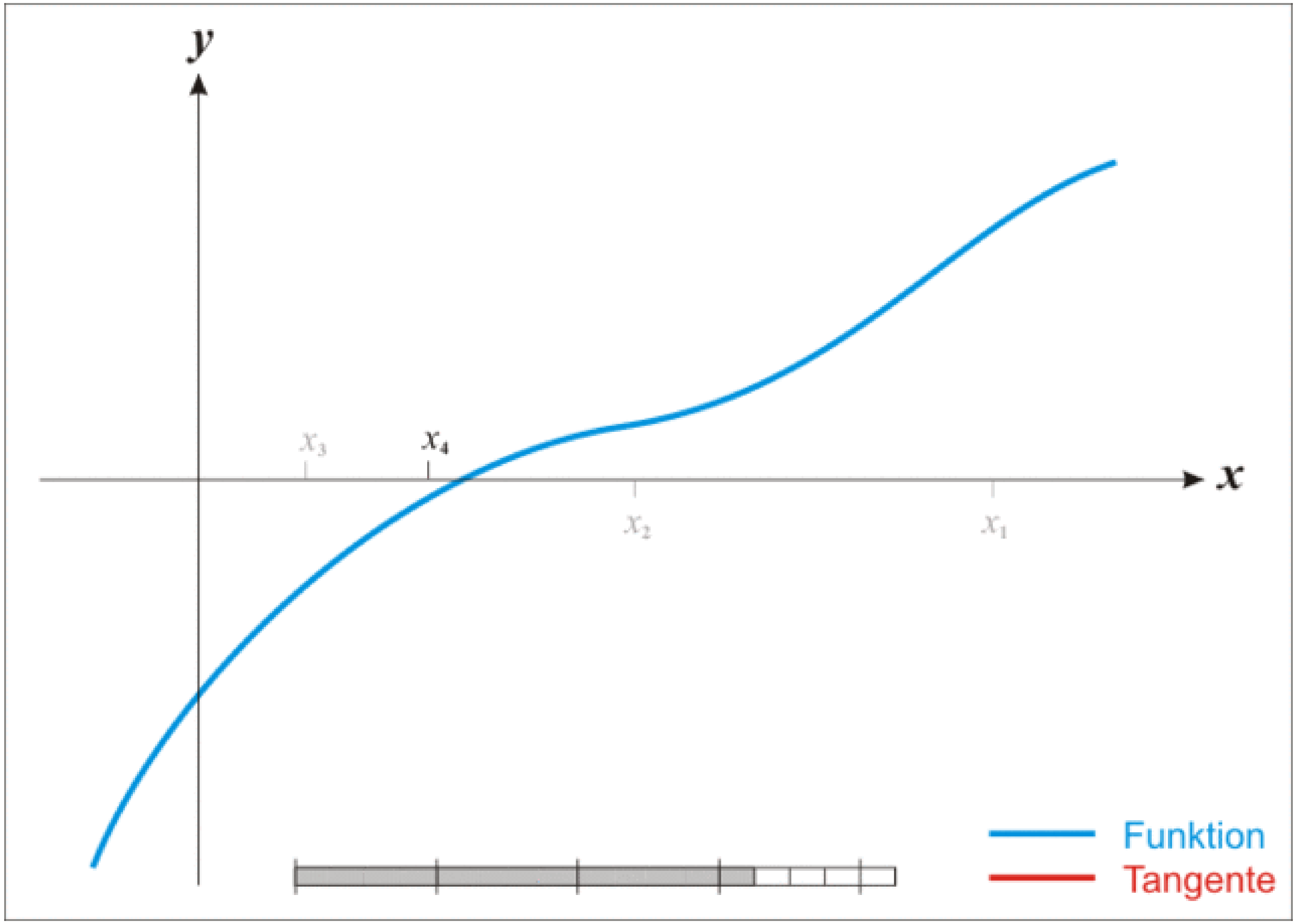
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



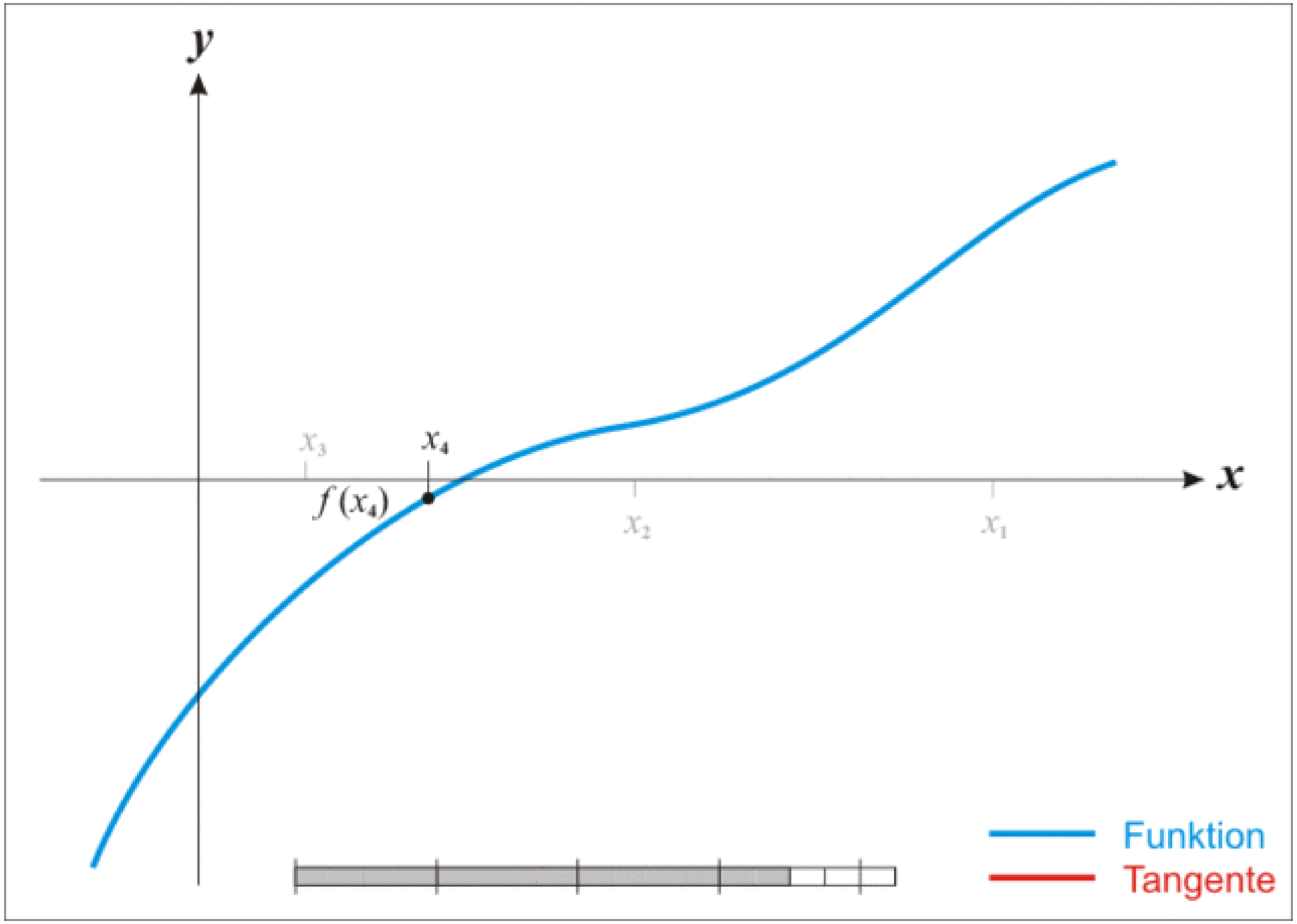
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



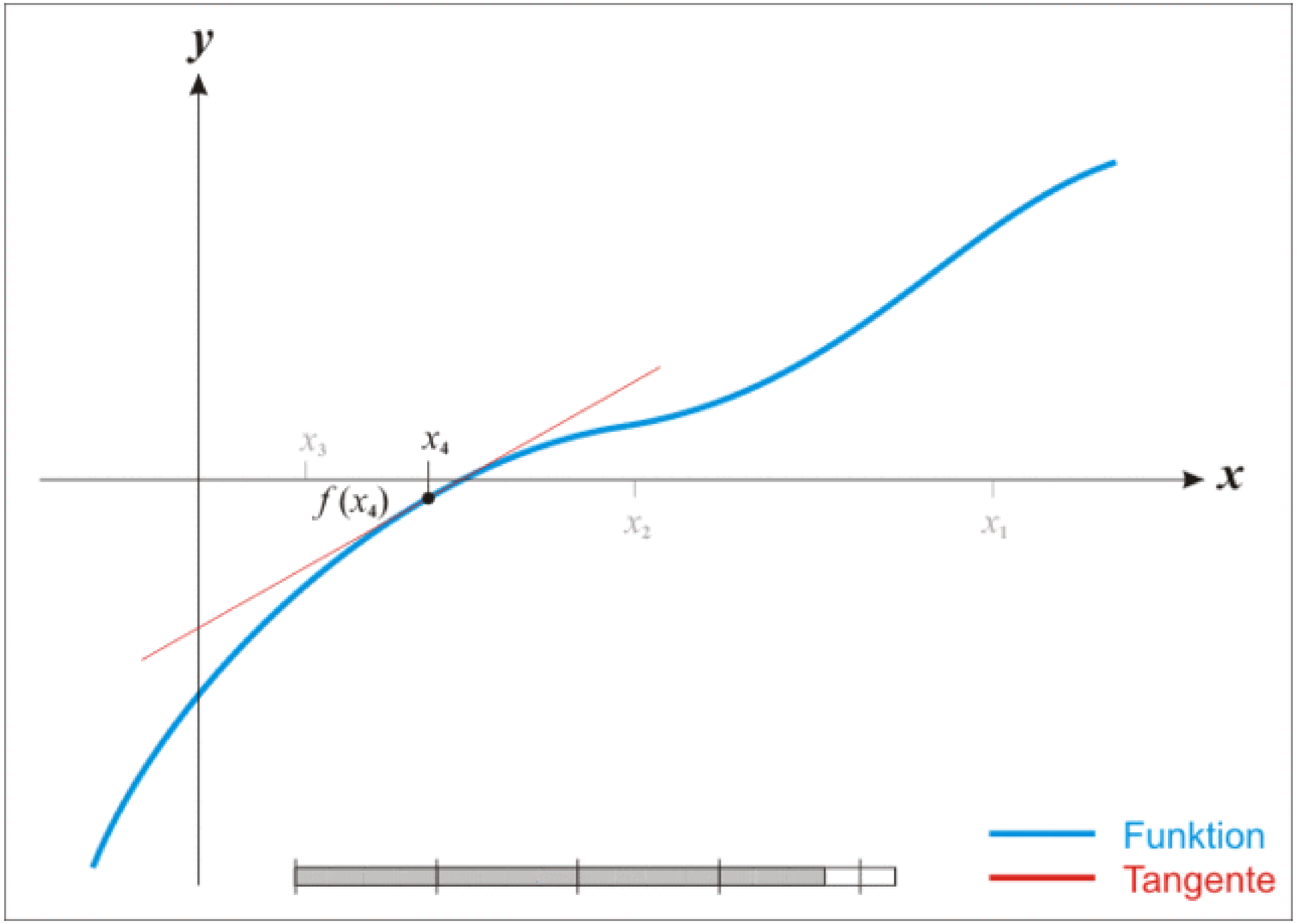
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



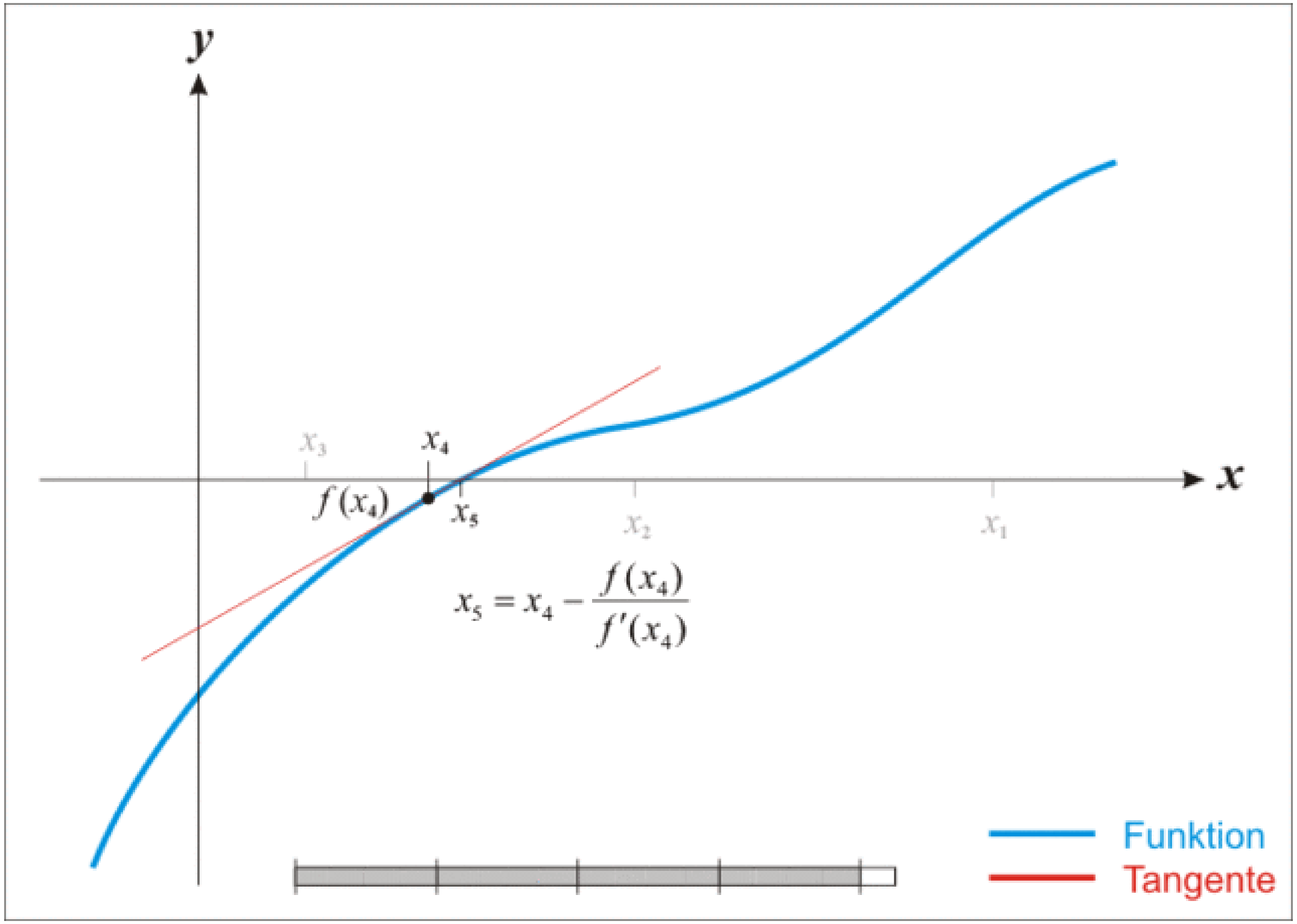
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



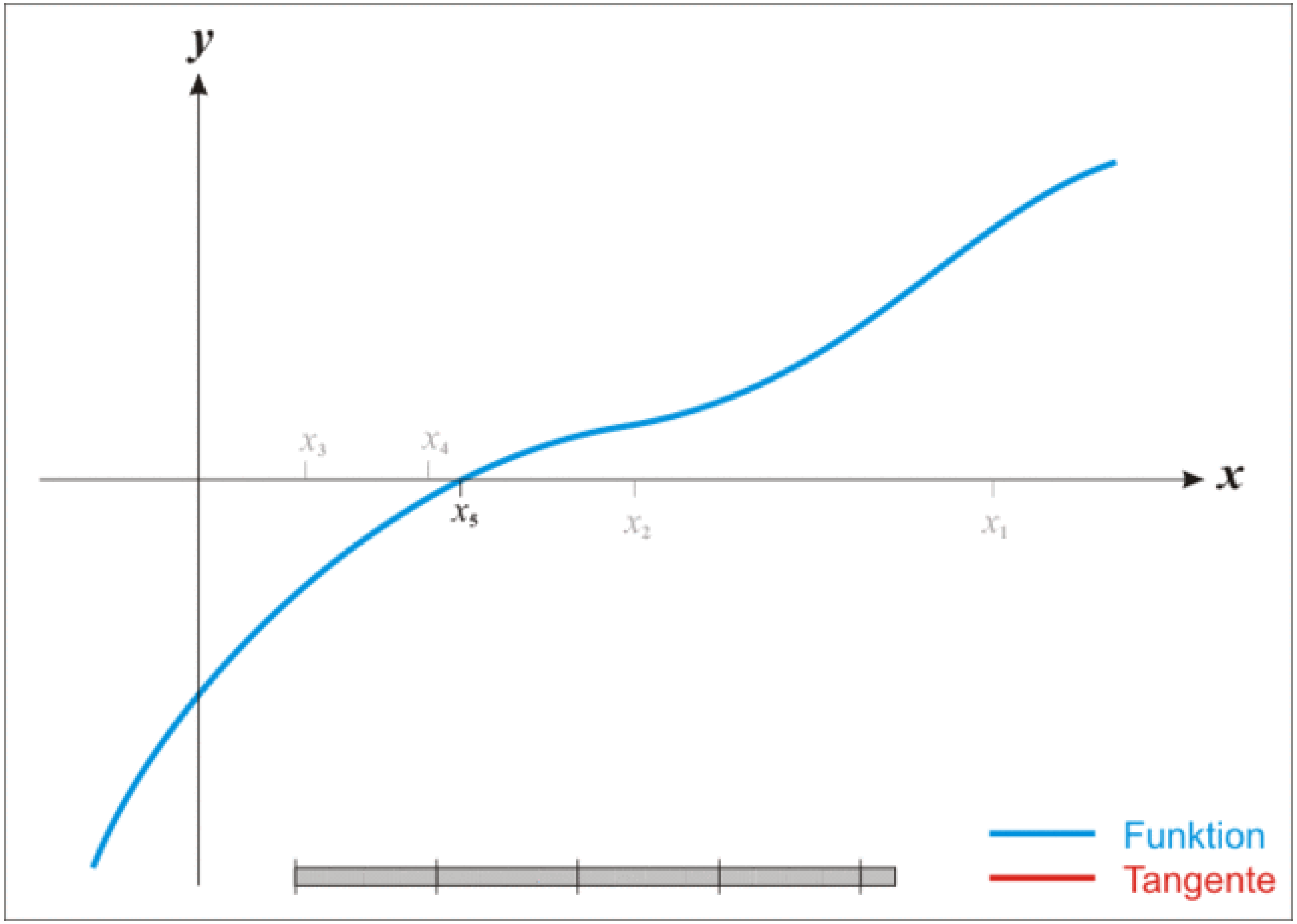
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif
by Ralf Pfeifer CC-BY-SA-3.0, via Wikimedia Commons



https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif



https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif

Applying Newton's Method to Finding Square Roots

- We can view the process of finding the square root of a number y as finding a solution to the equation:

$$x^2 = y$$

Applying Newton's Method to Finding Square Roots

- We can view the process of finding the square root of a number y as finding a solution to the equation:

$$x^2 - y = 0$$

Applying Newton's Method to Finding Square Roots

- Equivalently, we want to find a zero to the function:

$$f(x) = x^2 - y$$

Newton's Method

- Plugging in our function f:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Newton's Method

- Plugging in our function `f`:

$$x_{n+1} = x_n - \frac{x_n^2 - y}{2x_n}$$

Newton's Method

```
def abs(x: Double) = if (x > 0) x else 0 - x
def square(x: Double) = x * x
```

Newton's Method

- To encode Newton's Method as an application of generative recursion:
  - We need to choose an initial guess
  - We need to encode computation of the next guess from our current guess
  - We need to determine our base case

Newton's Method

- For square roots:
  - Our initial guess can be the parameter
  - Our base case is that our current guess falls within some tolerance of the true square root

Newton's Method

```
def next(guess: Double): Double =
  if (isGoodEnough(guess)) guess
  else next(guess - ((square(guess) - x) /
                     (2 * guess)))
```

Newton's Method

```
val epsilon = 1.0E-15

def isGoodEnough(guess: Double) =
  abs(square(guess) - x) <= epsilon
```

Newton's Method

```
def sqrt(x: Double) = {
  val epsilon = 1.0E-15

  def isGoodEnough(guess: Double) =
    abs(square(guess) - x) <= epsilon

  def next(guess: Double): Double =
    if (isGoodEnough(guess)) guess
    else next(guess - ((square(guess) - x) /
                       (2 * guess)))

  next(x)
}
```

Generalizing to an Arbitrary Function

```
def newtonsMethod(inverse: Double => Double)(y: Double) = {
  val epsilon = 1.0E-15
  val delta = 1.0E-9
  def f(x: Double) = inverse(x) - y

  def isGoodEnough(guess: Double) = abs(f(guess)) <= epsilon

  def fPrime(x: Double) = (f(x + delta) - f(x)) / delta

  def next(guess: Double): Double = {
    if (isGoodEnough(guess)) guess
    else next(guess - f(guess) / fPrime(guess))
  }
  next(y)
}
```

Generalizing to an Arbitrary Function

```
> newtonsMethod((x: Double) => x*x)(2)
res0: Double = 1.414213562373095

> newtonsMethod((x: Double) => x*x*x)(1000)
res1: Double = 10.0
```

Not All Applications of Newton's Method Terminate

- Consider:

$$f(x) = x^2 - x$$

$$f'(x) = 2x - 1$$

- An initial guess of 0.5 leads us to find the root of a tangent with slope zero (which has no root!)

Not All Applications of Newton's Method Terminate

```
newtonsMethod((x: Double) => x*x - x) ↦ ⊥
```

Design Recipe for Generative Recursion

- Data analysis and design
- Contract, purpose, header: Should now include some description of how the function works
- Examples: Include examples that illustrate how the function proceeds (not just input/output)

Design Recipe for Generative Recursion

- Template:
  - What is trivially solvable?
  - We new sub-problems do we generate?
  - How do we combine solutions to the sub-problems?
- Tests
- A termination argument

A Termination Argument

- With structural recursion, the computation follows the structure of the data
- Because immutable data has no cycles, the computation is certain to terminate
- With generative recursion, the sub-problems might be as large as the original problem
- Thus, we should include an explicit argument that the algorithm terminates