Dynamic Stochastic General Equilibrium Models

Understanding Deterministic Simulations (Perfect Foresight)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Willi Mutschler

Dynare Team

Deterministic Simulations

Recap

Deterministic Simulations

- perfect foresight = agents perfectly anticipate all future shocks and policy actions
- concretely, at period 1 agents
    - learn the value of all future shocks and/or policy changes
    - compute their optimal plans for all future periods
    - no need to adjust anything in periods 2 and later
- model behaves as if it were deterministic, i.e. no decision rules or uncertainty

Deterministic Simulations

- the *unknowns* that we search for are the trajectories of the variables (not a decision rule) given the dynamic model equations and initial values
- costs:
    - effect of future uncertainty is not taken into account (e.g. no precautionary motive)
    - unexpected shocks only impact (in period 1)
- advantages:
    - numerical solution can be computed exactly (up to rounding errors), contrarily to perturbation or global solution methods for rational expectations models
    - nonlinearities fully taken into account (e.g. occasionally binding constraints)

Deterministic Simulations

applications

- initial model assessment, first glance at the propagation of shocks
- certain and predictable structural changes (e.g. taxes, new currency)
- long-run simulations (from one steady-state to another one)
- large models
- large shocks
- kinks and nonlinearities

Examples

Two-Country New-Keynesian Model with Zero-Lower-Bound on Interest Rates

Common Model Equations

nk2co_common.mod

Two-Country New-Keynesian Model with Zero-Lower-Bound on Interest Rates

Temporary Monetary Policy Shock
(Surprise)

nk2co_temp_monpol_surprise.mod

Two-Country New-Keynesian Model with Zero-Lower-Bound on Interest Rates

Temporary Monetary Policy Shock (Pre-Announced)

nk2co_temp_monpol_announced.mod

Two-Country New-Keynesian Model with Zero-Lower-Bound on Interest Rates

Permanent Increase Inflation Target
(Surprise)

nk2co_perm_infltarget_surprise.mod

Two-Country New-Keynesian Model with Zero-Lower-Bound on Interest Rates

Permanent Increase Income Tax
(Pre-Announced)

nk2co_perm_tax_announced.mod

Dynare Specifics

Summary of Commands

`initval`:	for the initial steady state (followed by steady)
`endval`:	for the terminal steady state (followed by steady)
`histval`:	for initial or terminal conditions out of steady state
`shocks`:	for shocks along the simulation path
`perfect_foresight_setup`:	prepare the simulation
`perfect_foresight_solver`:	compute the simulation

Under The Hood

- paths for exogenous and endogenous variables are stored in two MATLAB/Octave matrices:

$$\text{oo\_.endo\_simul} = \begin{pmatrix} y_0 & y_1 & \cdots & y_T & y_{T+1} \end{pmatrix}$$

$$\text{oo\_.exo\_simul'} = \begin{pmatrix} \boxtimes & u_1 & \ldots & u_T & \boxtimes \end{pmatrix}$$

- for historical reasons dates are in
    - columns in oo_.endo_simul
    - lines in oo_.exo_simul (hence the transpose ' above)

Under The Hood

`perfect_foresight_setup`

- initializes those matrices, given the `shocks`, `initval`, `endval` and `histval` blocks:
  - y_0, y_{T+1} and $u_1 \ldots u_T$ are the constraints of the problem
  - $y_1 \ldots y_T$ are the initial guess for the Newton algorithm

`perfect_foresight_solver`

- replaces $y_1 \ldots y_T$ in oo_.endo_simul by the solution

The Algorithm

General DSGE Framework

- deterministic, perfect foresight, case:

$$f(y_{t+1}, y_t, y_{t-1}, u_t) = 0$$

y: vector of endogenous variables

u: vector of exogenous shocks

- identification rule: as many endogenous (y) as equations (f)

More Than One Lead/Lag?

- can be transformed in the form with one lead and one lag using *auxiliary* variables:
- for example, if there is a variable with two leads x_{t+2}:
    - create a new auxiliary variable a
    - replace all occurrences of x_{t+2} by a_{t+1}
    - add a new equation: $a_t = x_{t+1}$
- symmetric process for variables with more than one lag
- with future uncertainty, the transformation is more elaborate (but still possible) on variables with leads
- transformation done automatically by Dynare

Two-Boundary Value Problem

stacked system for a perfect foresight simulation over T periods:

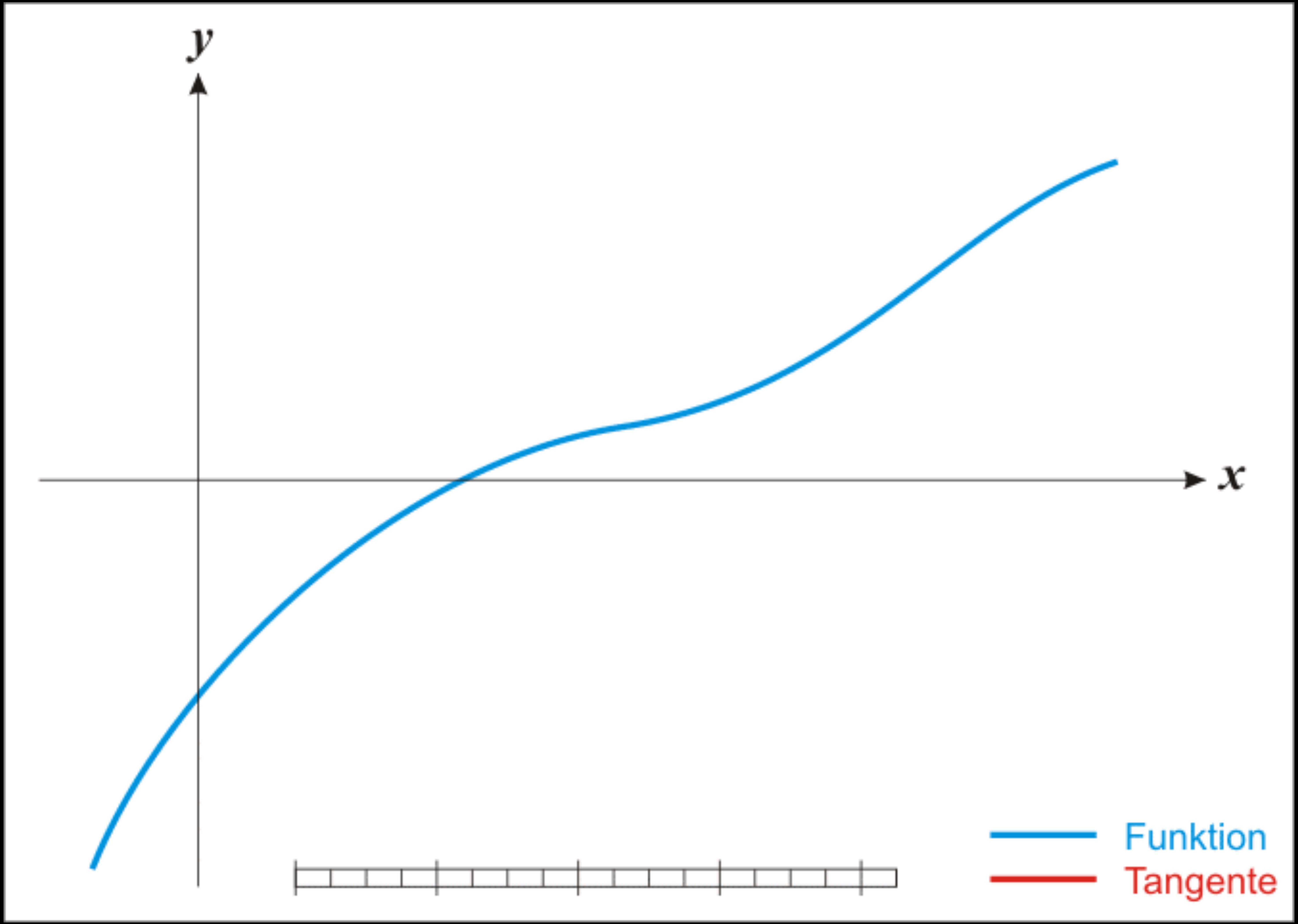
$$F(Y) = \begin{cases} f(y_2, y_1, y_0, u_1) = 0 \\ f(y_3, y_2, y_1, u_2) = 0 \\ \vdots \\ f(y_{T+1}, y_T, y_{T-1}, u_T) = 0 \end{cases} \qquad \text{for } y_0 \text{ and } y_{T+1} \text{ given}$$

where $Y = \begin{bmatrix} y_1' & y_2' & \dots & y_T' \end{bmatrix}'$ and y_0, y_{T+1}, $u_1 \ldots u_T$ are implicit

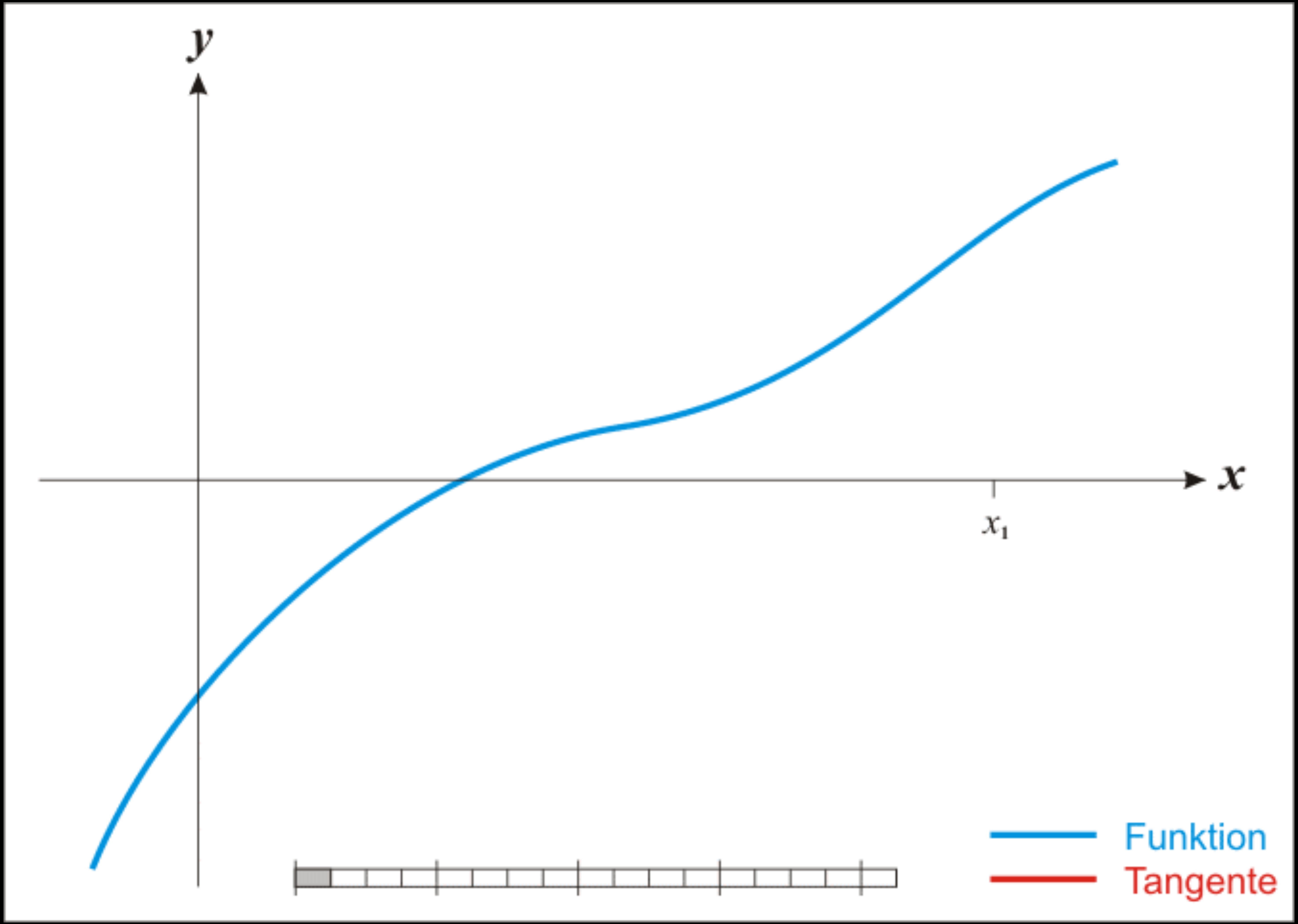
goal: find a trajectory Y, i.e. values, for $y_1, y_2, \ldots, y_T$ given y_0, y_{T+1}, $u_1 \ldots u_T$

solution: Newton-type iterations

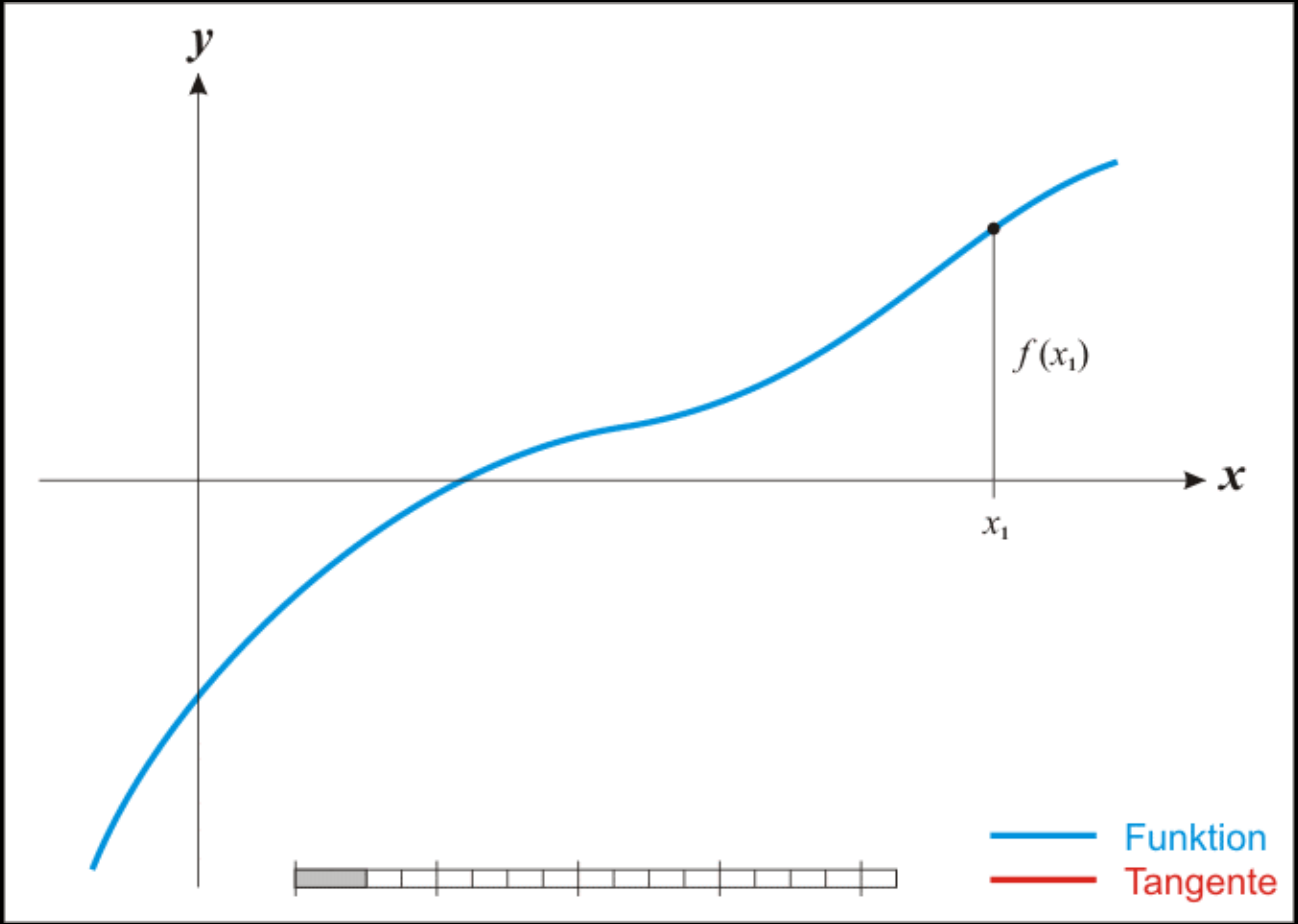
Newton Method (1)



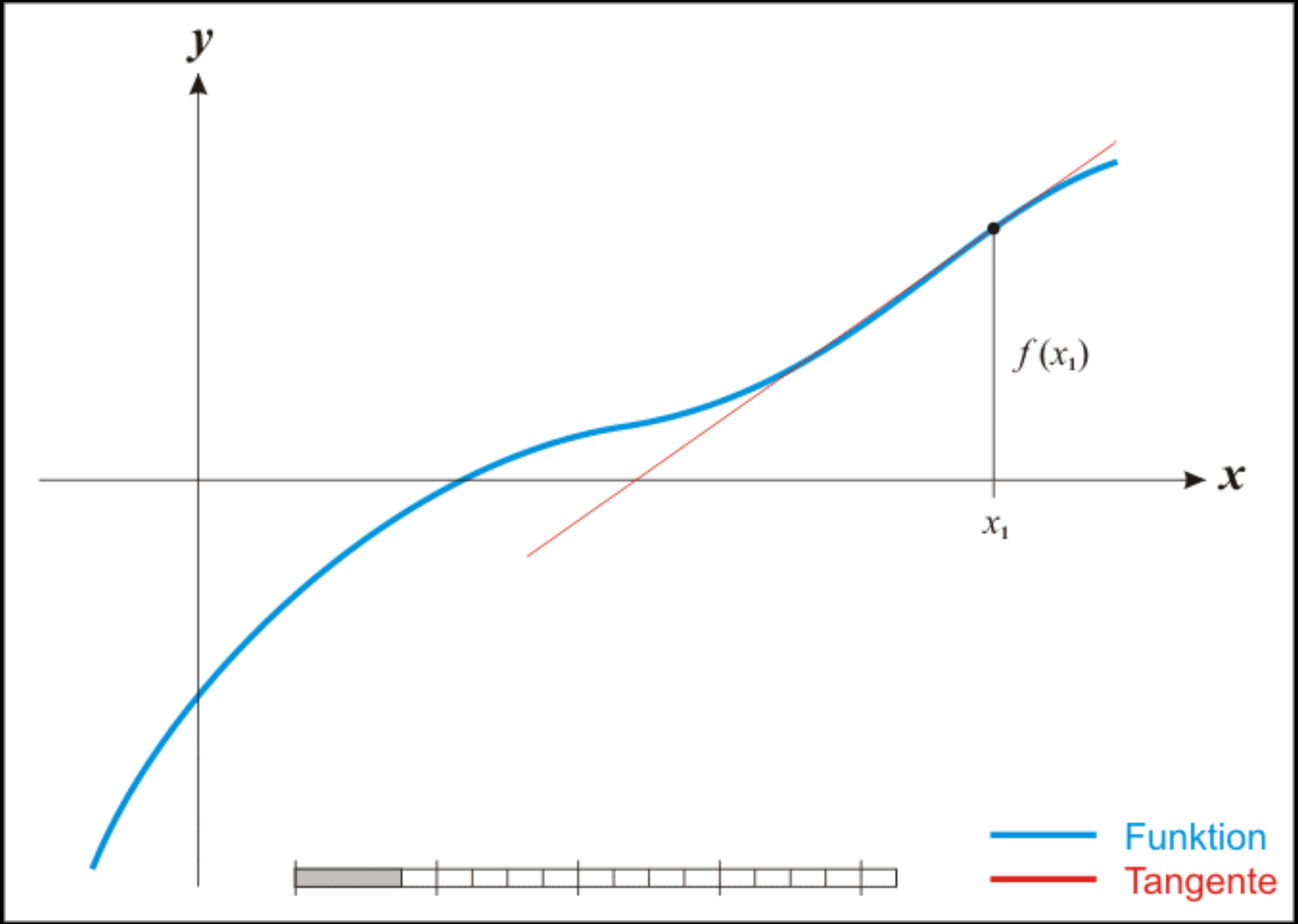
Newton Method (2)



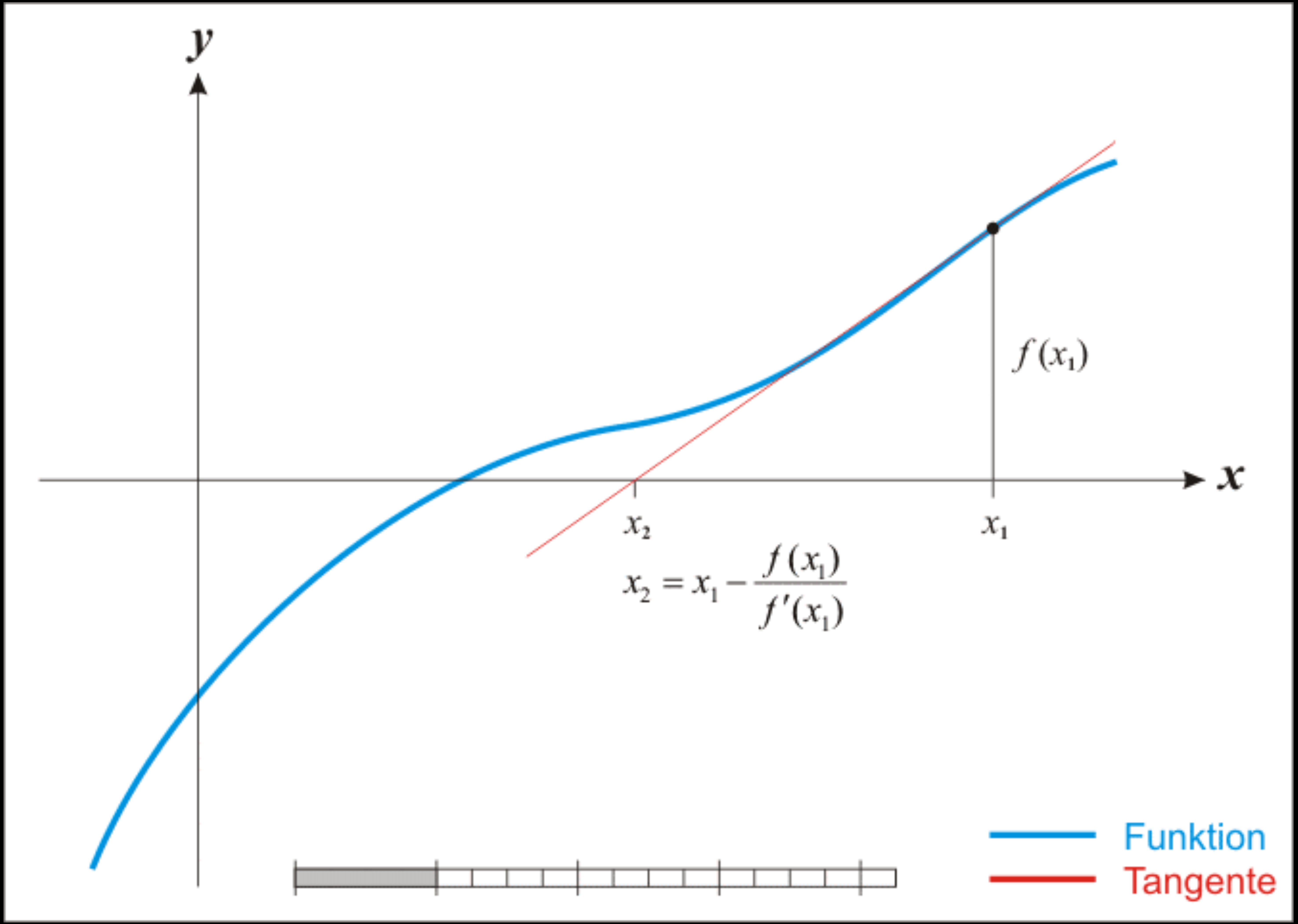
Newton Method (3)



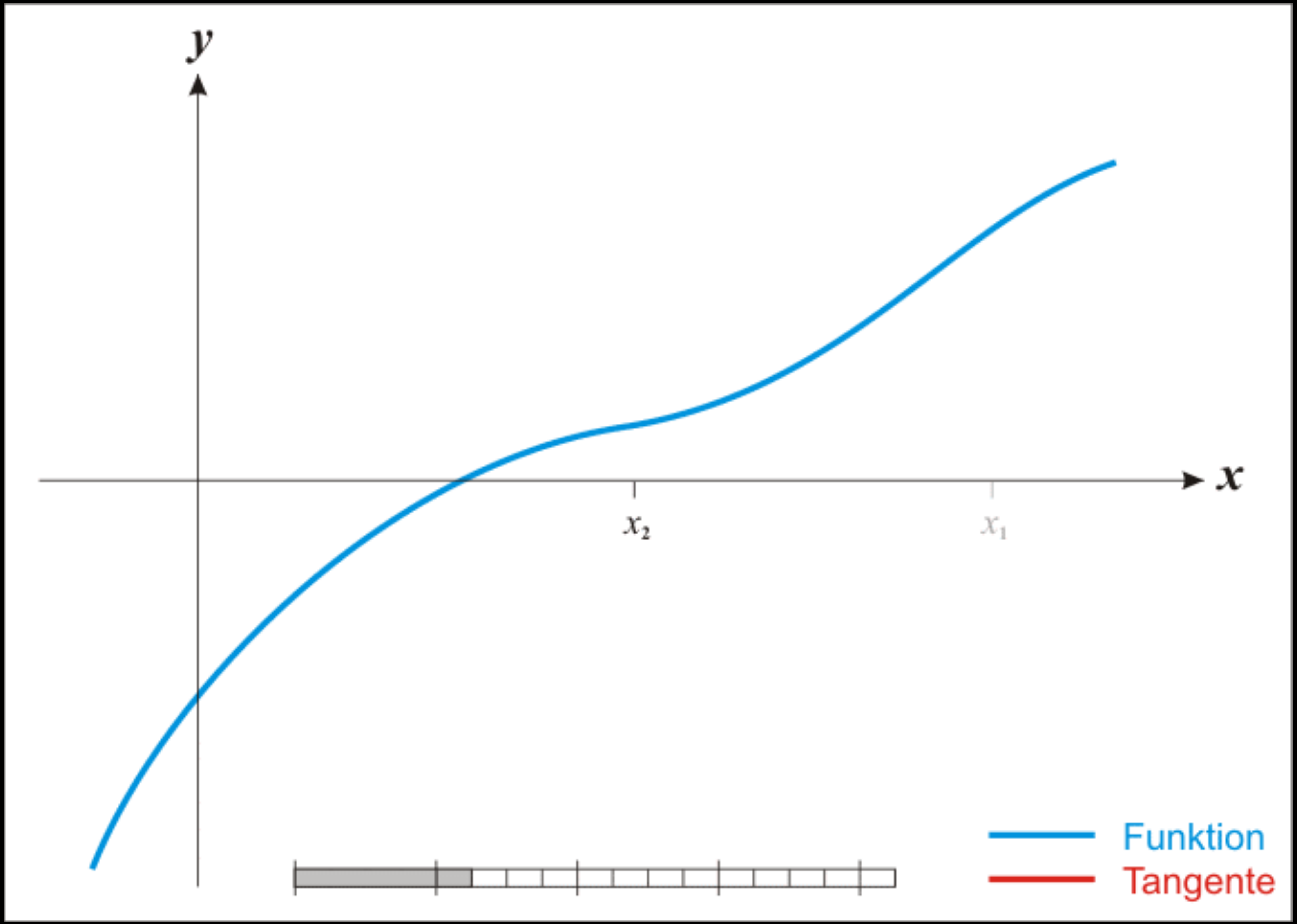
Newton Method (4)



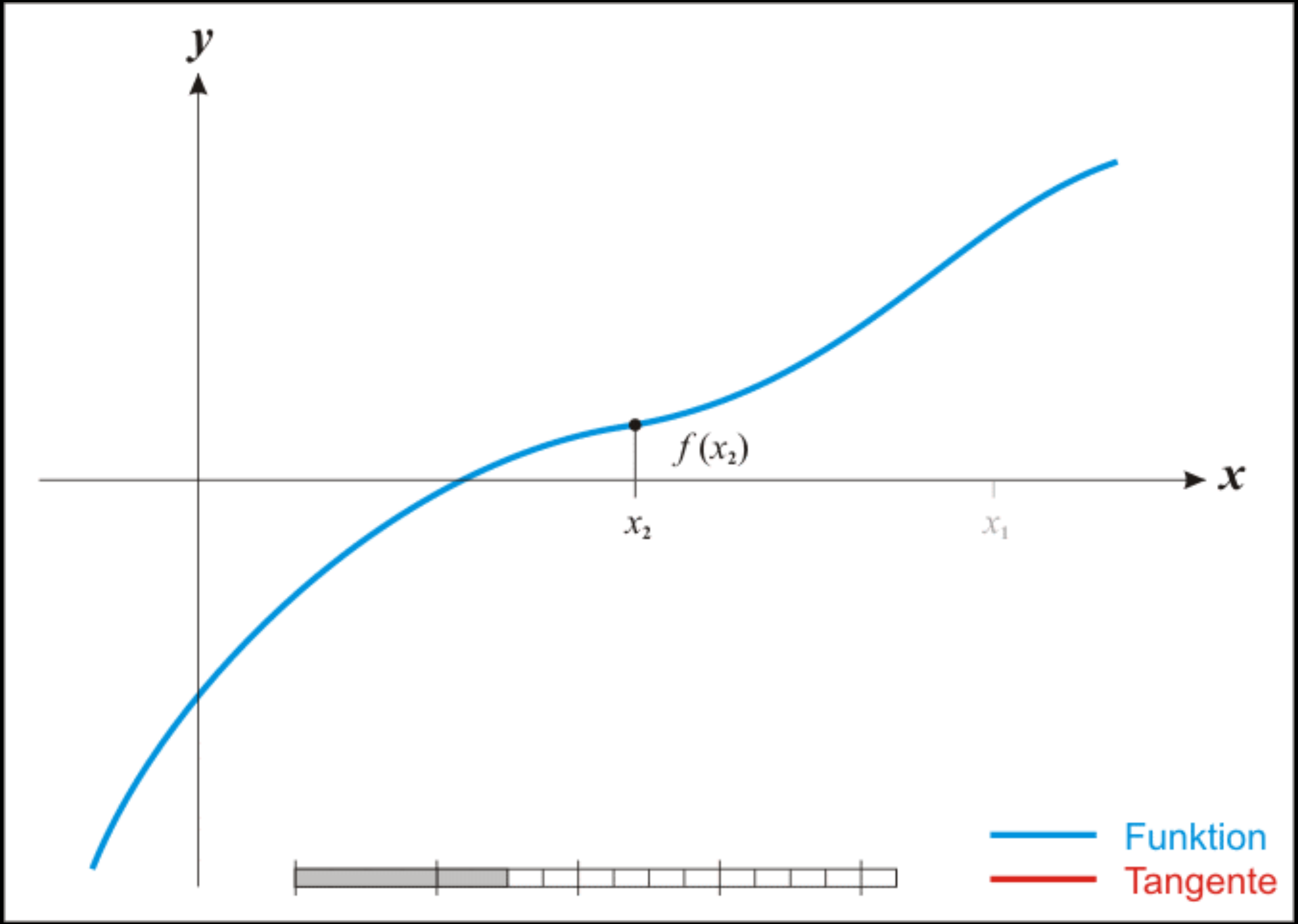
Newton Method (5)



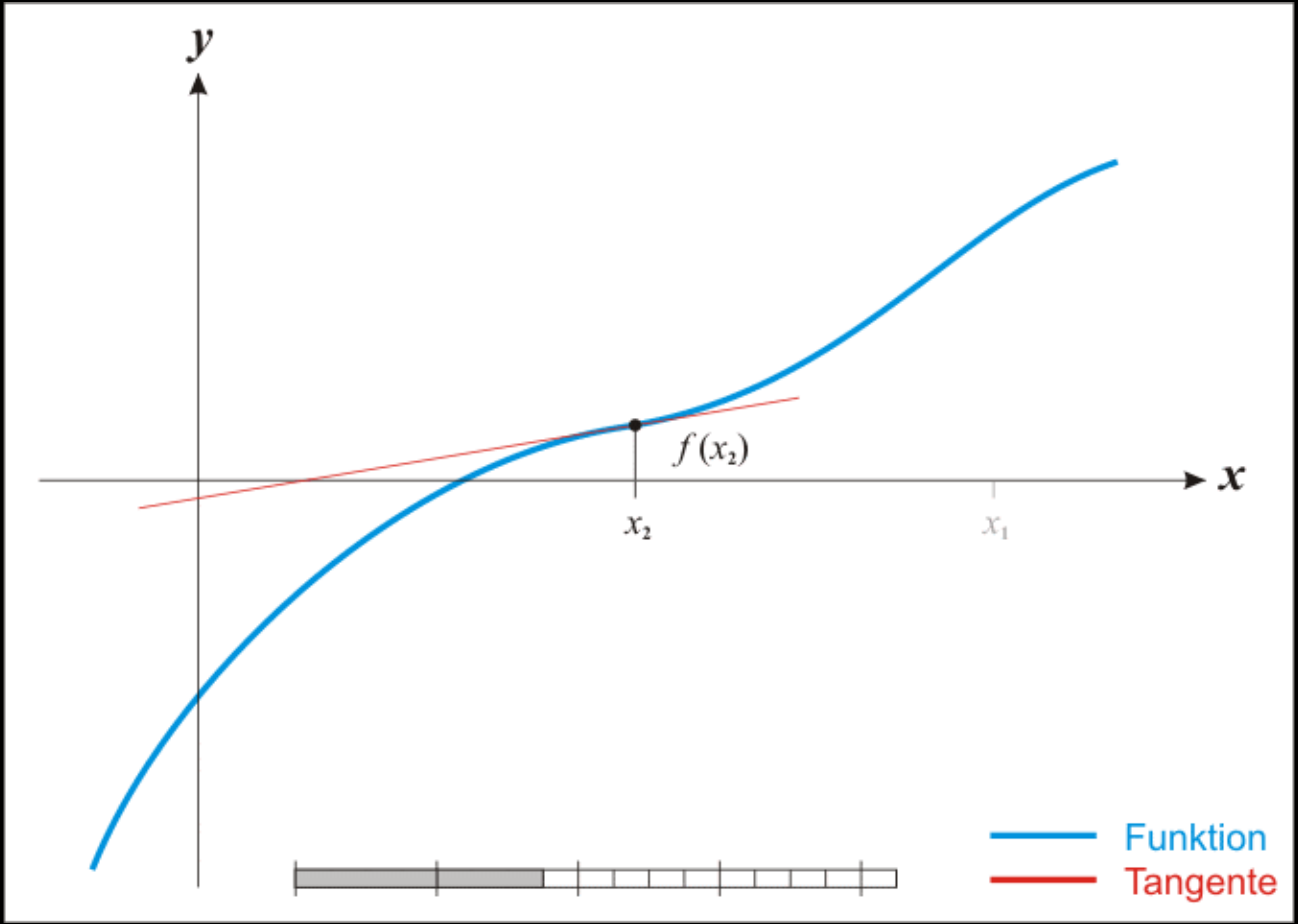
Newton Method (6)



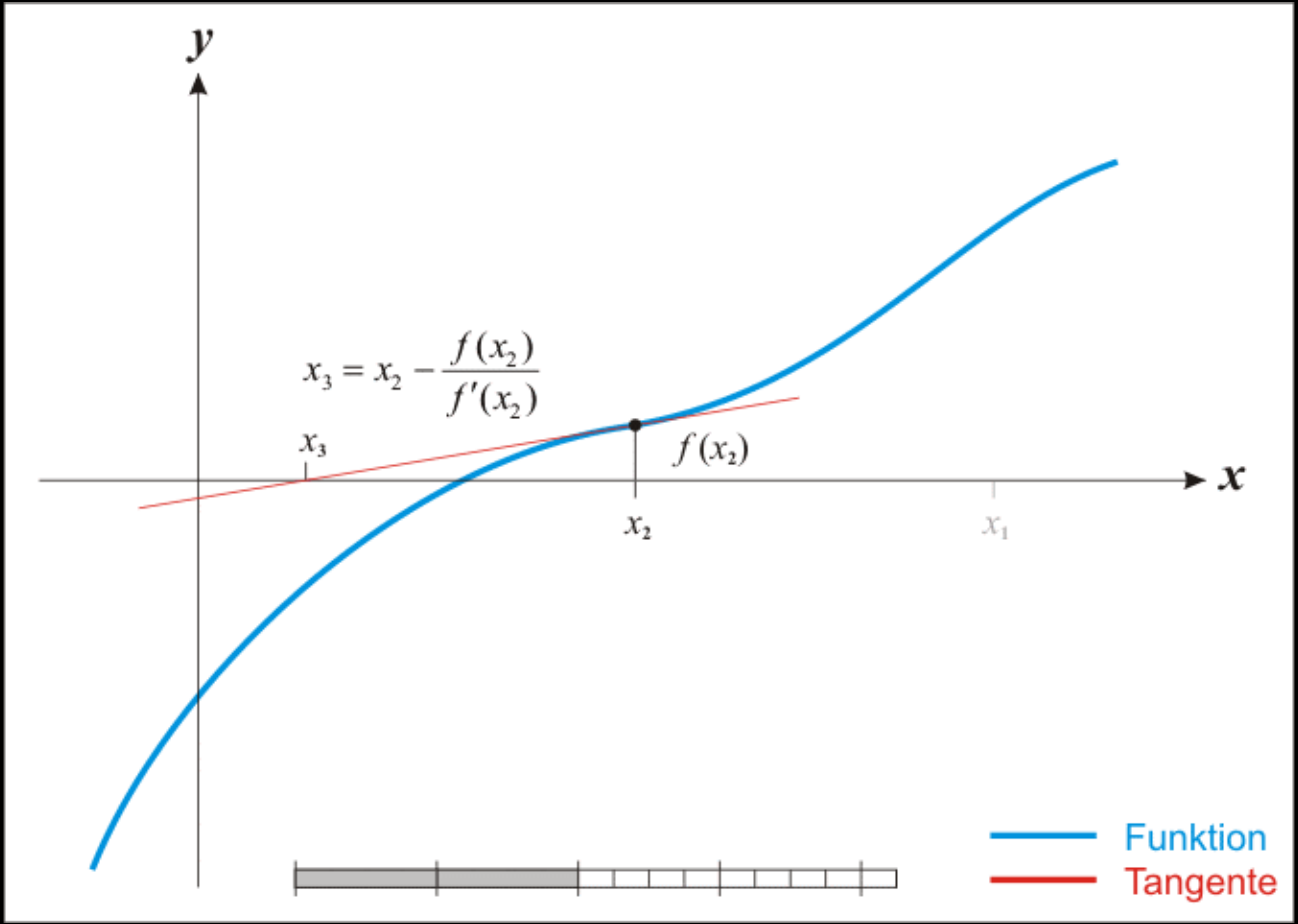
Newton Method (7)



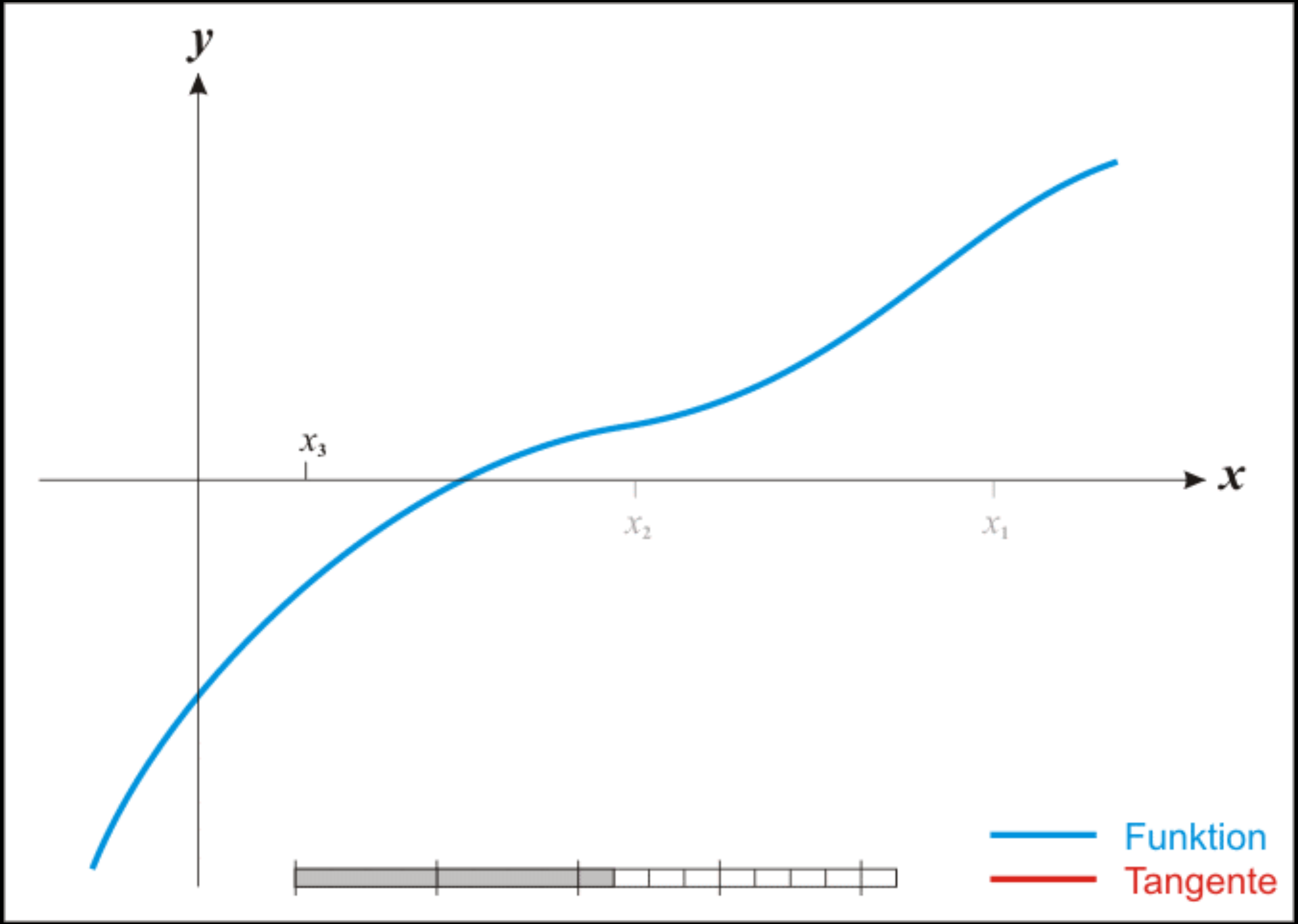
Newton Method (8)



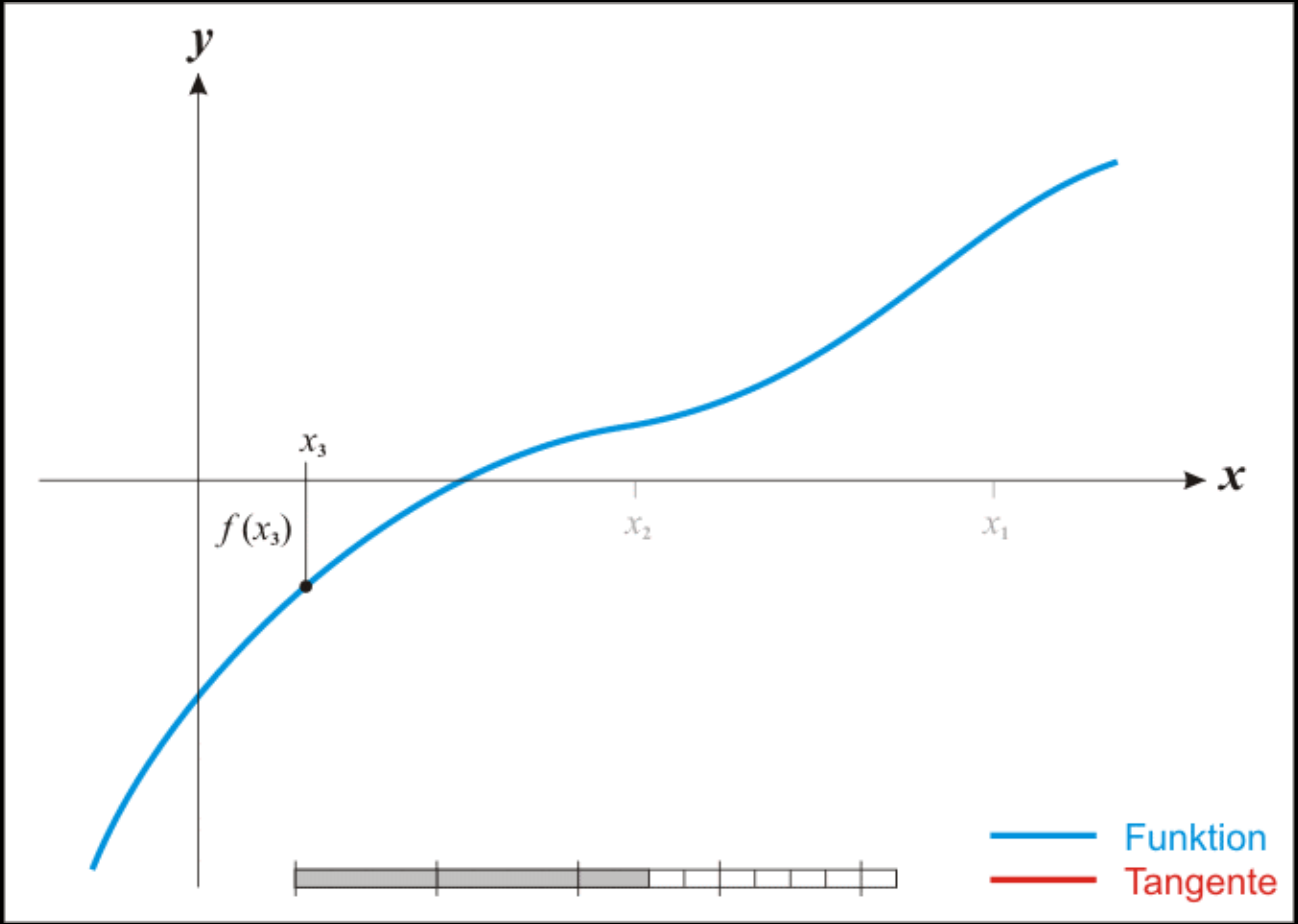
Newton Method (9)



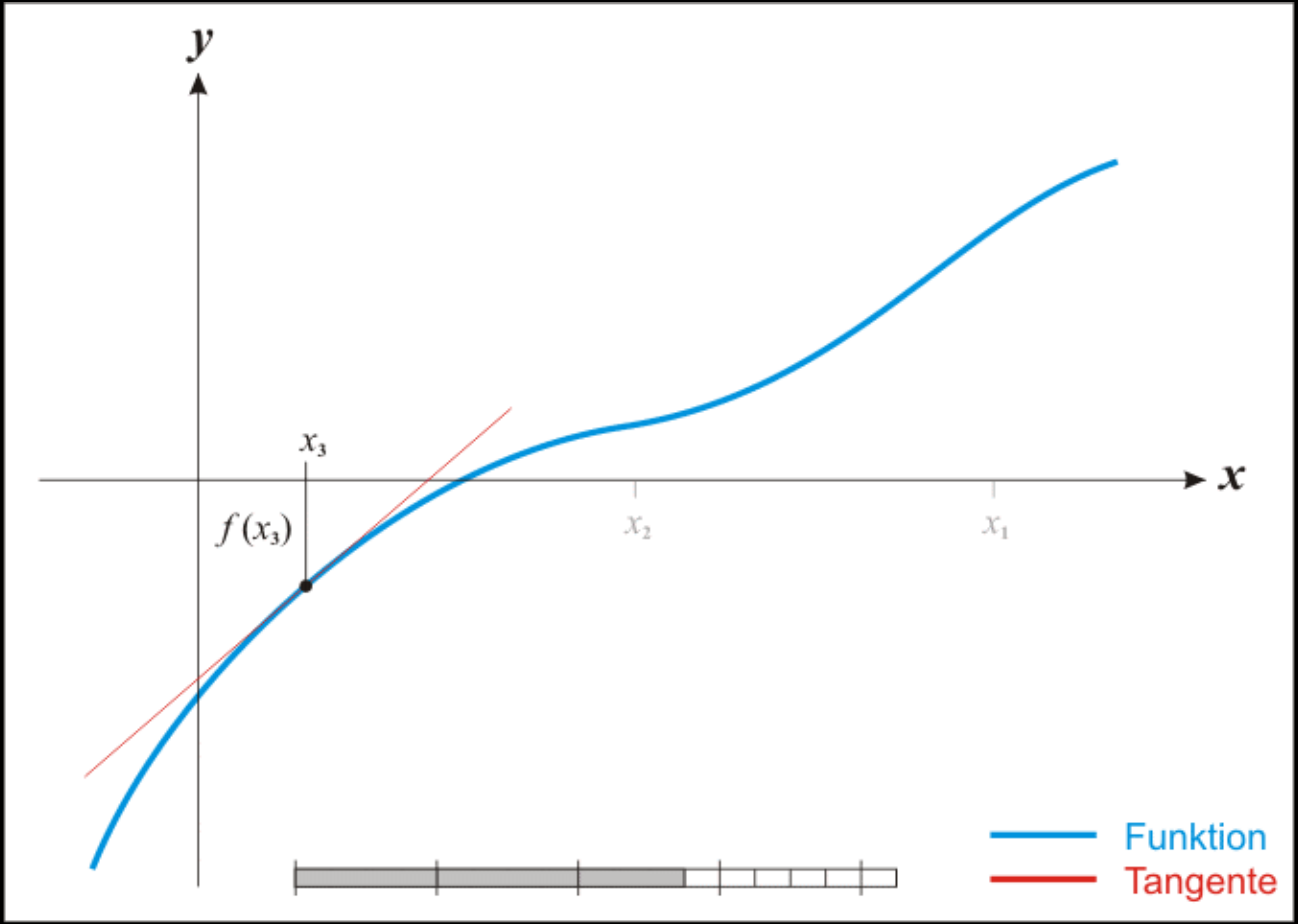
Newton Method (10)



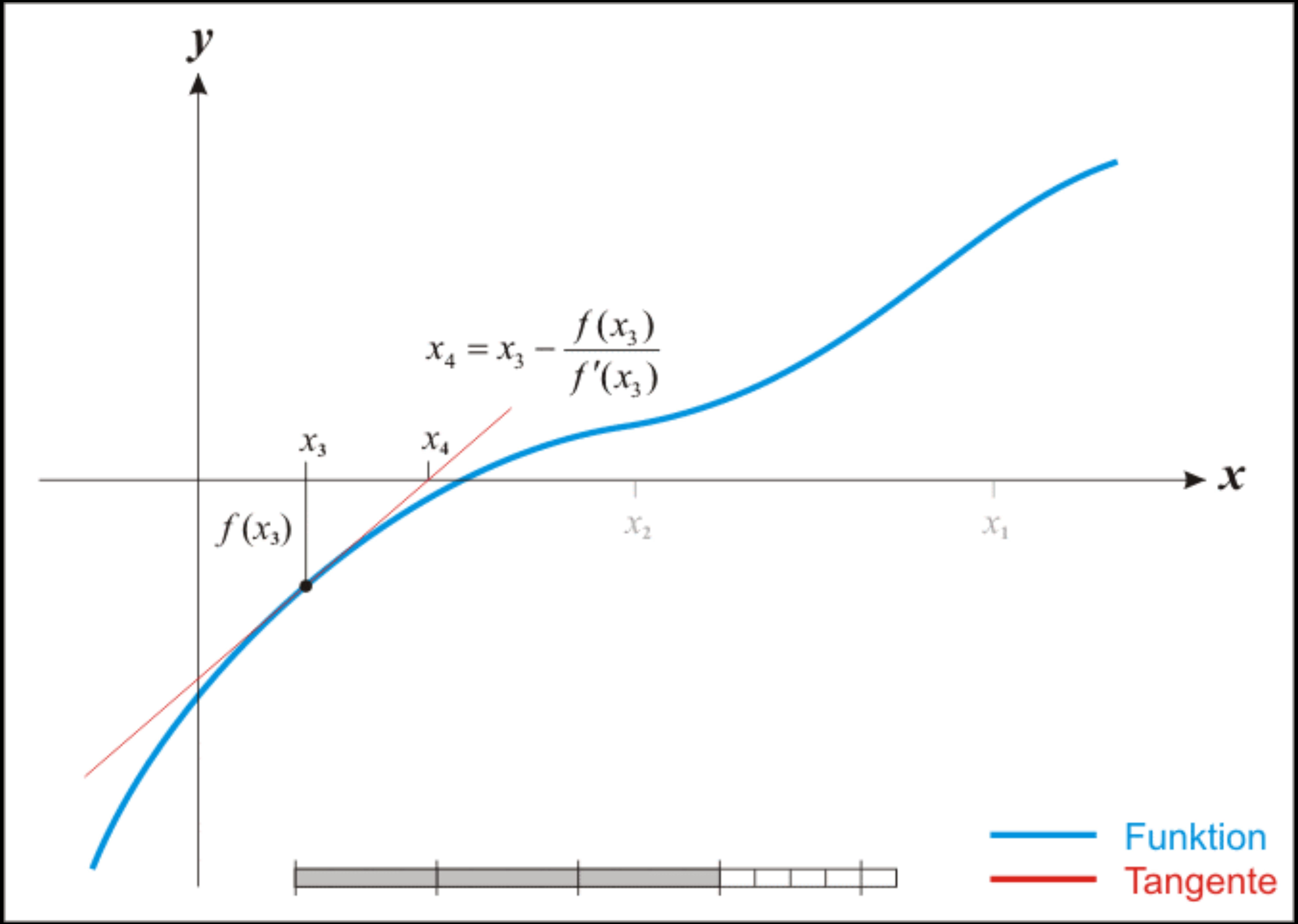
Newton Method (11)



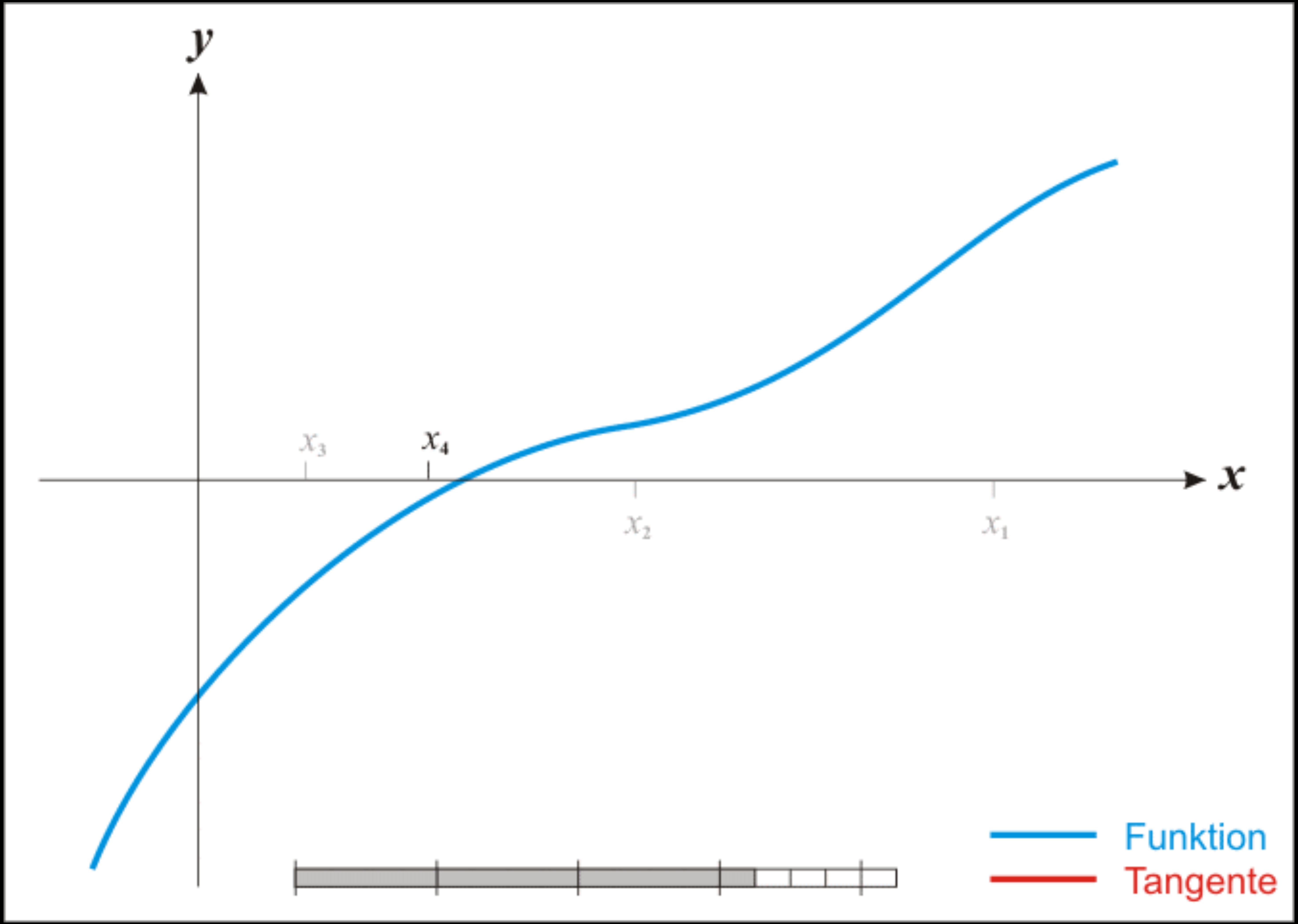
Newton Method (12)



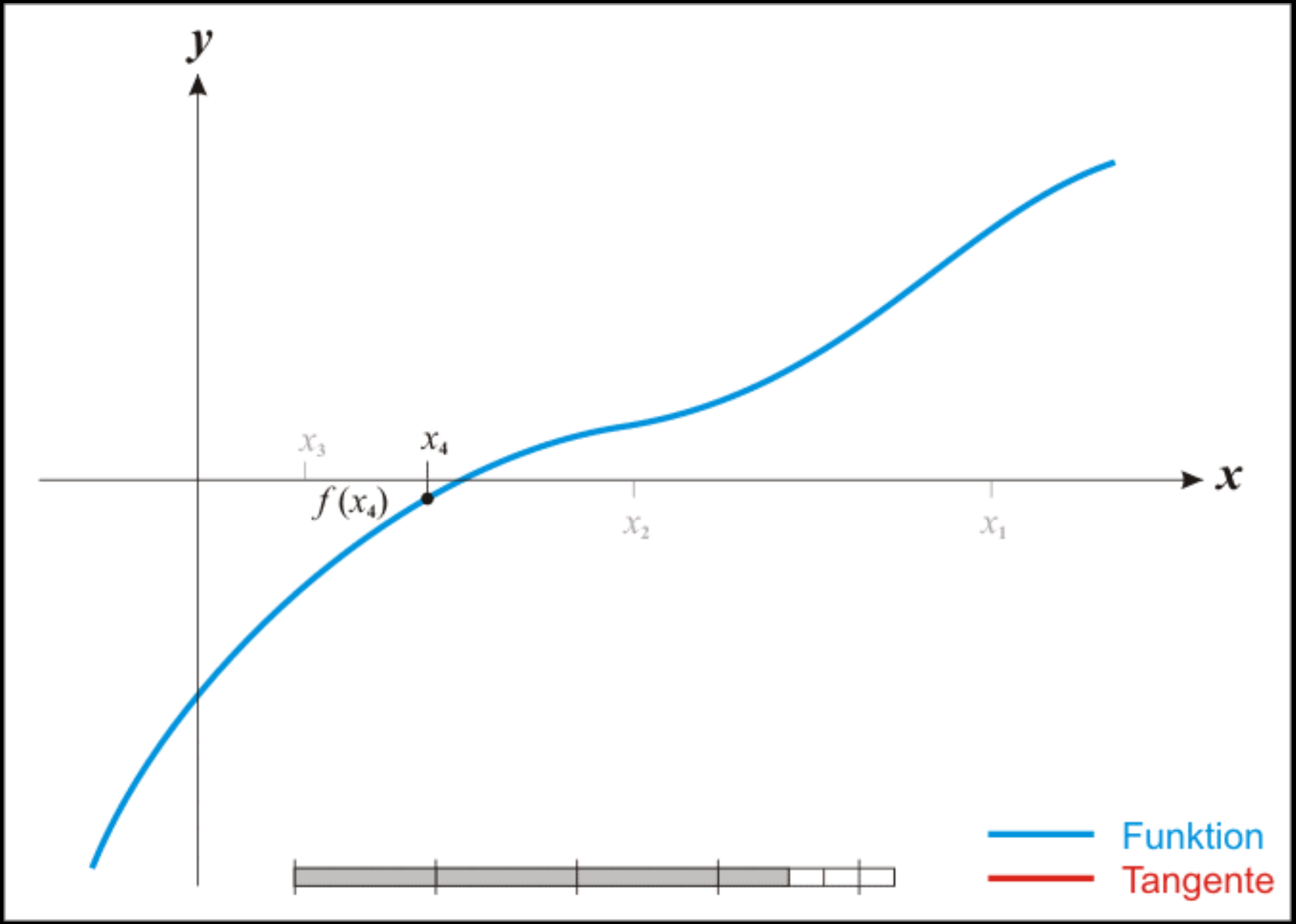
Newton Method (13)



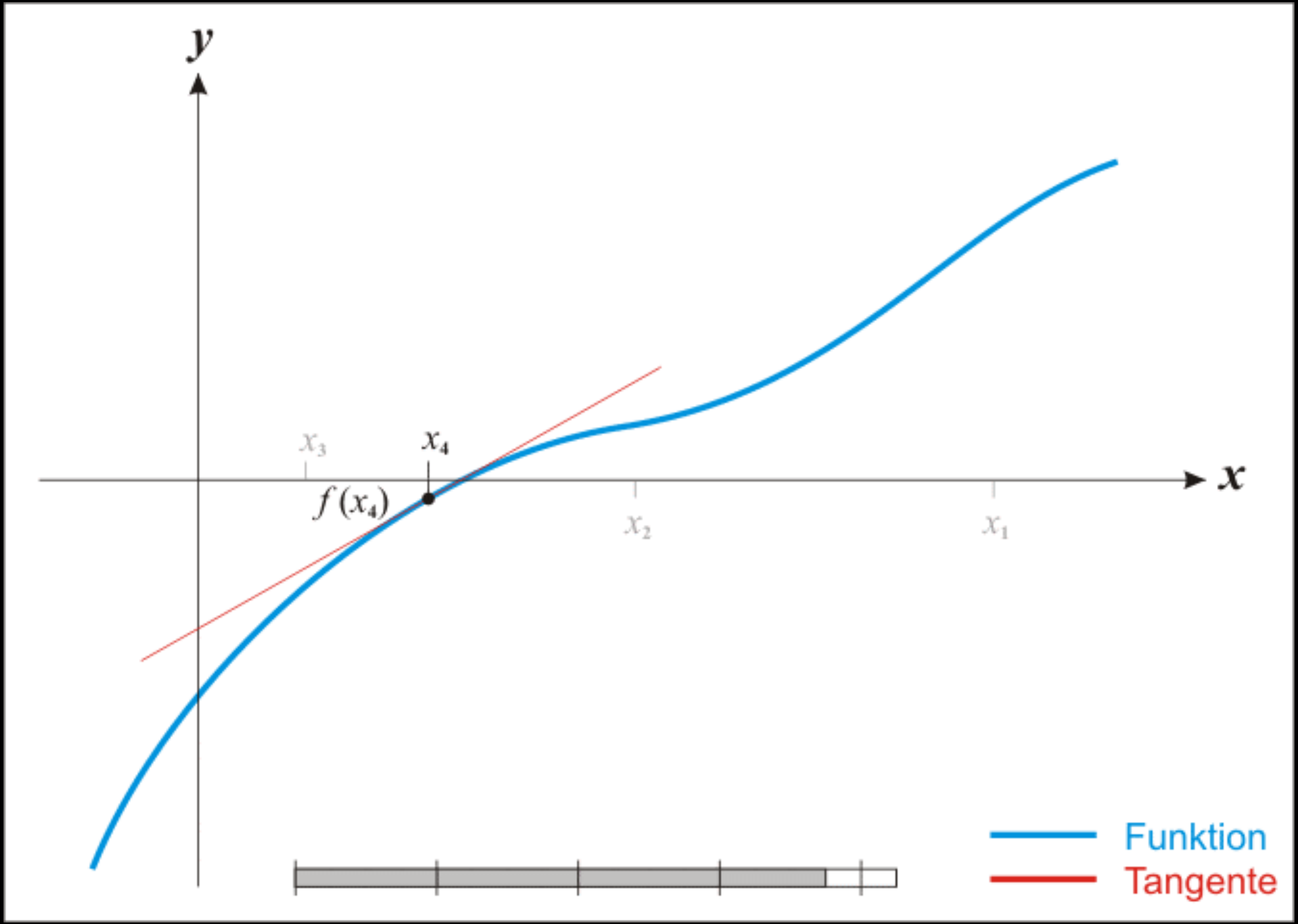
Newton Method (14)



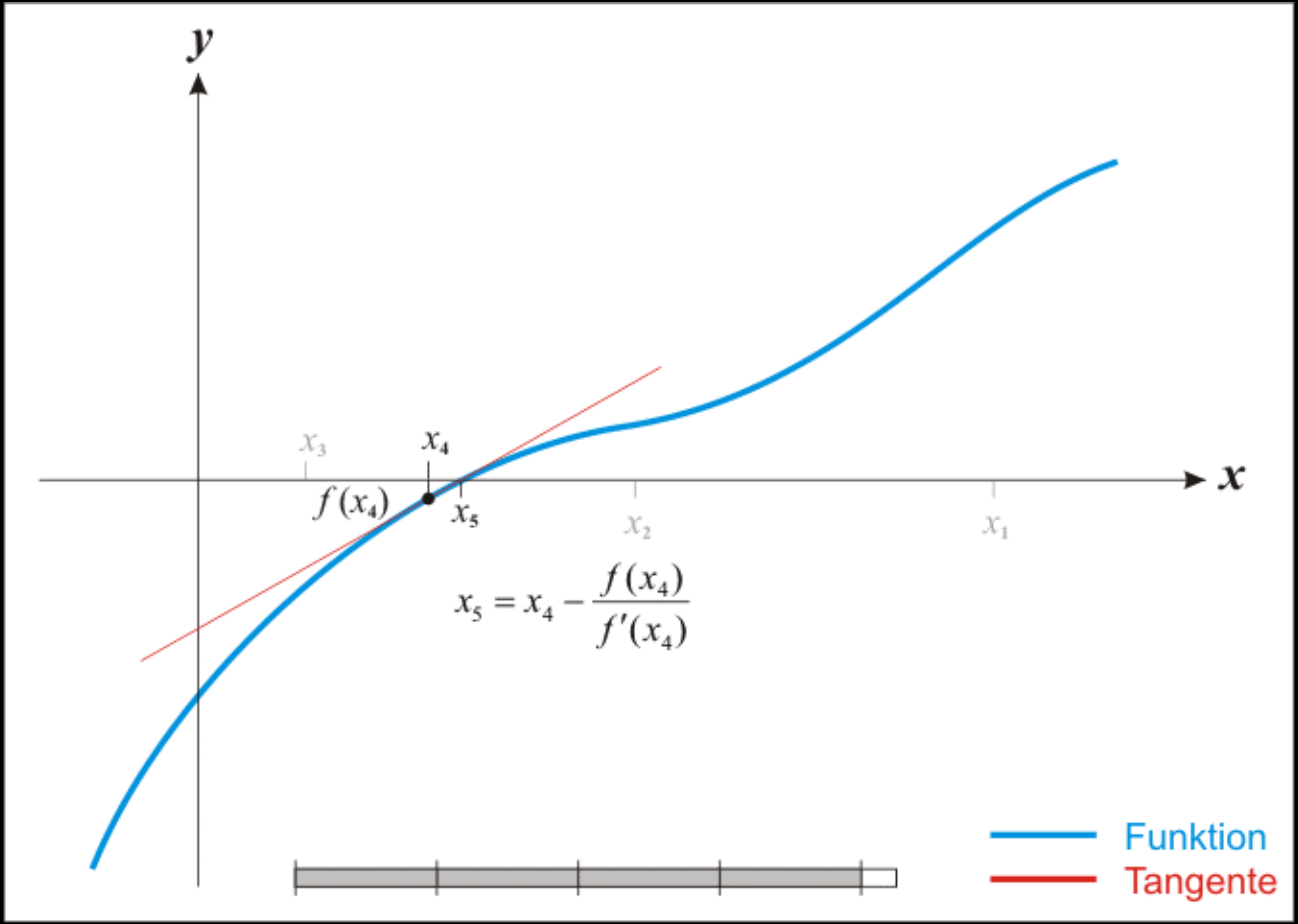
Newton Method (15)



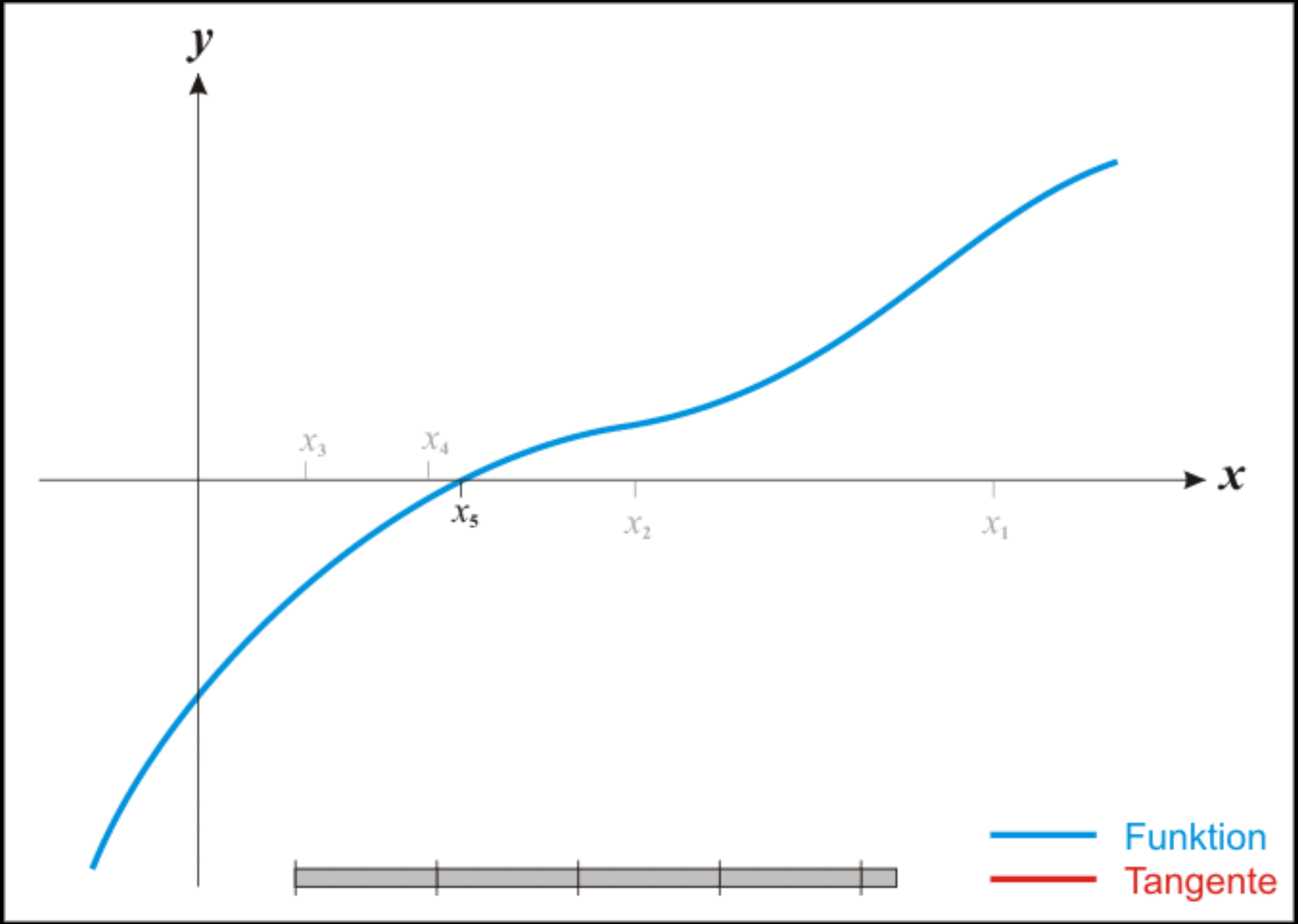
Newton Method (16)



Newton Method (17)



Newton Method (18)



The Perfect Foresight Algorithm

- start from an initial guess $Y^{(0)}$ (incorporating the simulation scenario)

- iterate according to Newton algorithm

- updated solutions $Y^{(k+1)}$ are obtained by solving a linear system:

$$F(Y^{(k)}) + \left[\frac{\partial F}{\partial Y}\right]\left(Y^{(k+1)} - Y^{(k)}\right) = 0$$

$$\Leftrightarrow \left(Y^{(k+1)} - Y^{(k)}\right) = -\left[\frac{\partial F}{\partial Y}\right]^{-1} F(Y^{(k)})$$

- terminal condition: $||Y^{(k+1)} - Y^{(k)}|| < \varepsilon_Y$ or $||F(Y^{(k)})|| < \varepsilon_F$

- convergence may never happen if function is ill-behaved or initial guess $Y^{(0)}$ too far from a solution (abort infinite loops by setting a maximum number of iterations)

Controlling Newton Algorithm From Dynare

options to `perfect_foresight_solver` can be used to control the Newton algorithm:

- `maxit`: Maximum number of iterations before aborting (default: 50)
- `tolf`: Convergence criterion based on function value (ε_F) (default: 10^{-5})
- `tolx`: Convergence criterion based on change in the function argument (ε_Y) (default: 10^{-5})
- `stack_solve_algo`: select between the different flavors of Newton algorithms

Initial Guess

- Newton algorithm needs an initial guess $Y^{(0)} = [y_1^{(0)'} \; \dots \; y_T^{(0)'}]$
- by default, if there is no `endval` block, it is the steady state as specified by `initval` (repeated for all simulations periods)
- if there is an `endval` block, then it is the final steady state declared within this block
- possibility of customizing this default by manipulating oo_.endo_simul
  after `perfect_foresight_setup`
  but before (!) `perfect_foresight_solver`

Approximating Infinite-Horizon Problems

- technically we numerically compute trajectories over a *finite* number of periods T
- what about an *infinite*-horizon problem (e.g. return to steady-state) $T \to \infty$?
  - one option consists in computing a recursive policy function (as with perturbation methods)
  - but this is challenging, Dynare does not do that
- easier way:
  - approximate the solution by a finite-horizon problem with T *large enough*
  - drawback: solution is specific to a given sequence of shocks and not generic

Jacobian

Shape Of Jacobian

$$\frac{\partial F}{\partial Y} = \begin{pmatrix} B_1 & C_1 & & & & & \\ A_2 & B_2 & C_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & A_t & B_t & C_t & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & A_{T-1} & B_{T-1} & C_{T-1} \\ & & & & & A_T & B_T \end{pmatrix}$$

$$A_s = \frac{\partial f}{\partial y_{t-1}}(y_{s+1}, y_s, y_{s-1}) \qquad B_s = \frac{\partial f}{\partial y_t}(y_{s+1}, y_s, y_{s-1}) \qquad C_s = \frac{\partial f}{\partial y_{t+1}}(y_{s+1}, y_s, y_{s-1})$$

Shape Of Jacobian

- the Jacobian can be very large: for a simulation over T periods of a model with n endogenous variables, it is a matrix of dimension $nT \times nT$

- three alternative ways of dealing with the large problem size:

    - exploit the particular structure of the Jacobian using a technique developped by Laffargue, Boucekkine and Juillard (was the default method in Dynare ≤ 4.2)
    `stack_solve_algo=6`

    - handle the Jacobian as one large, sparse, matrix (now the default method)
    `stack_solve_algo=0`

    - block decomposition, which is a divide-and-conquer method (can actually be combined with one of the previous two methods)

Sparse Matrices

- consider the following matrix with most elements equal to zero:

$$A = \begin{pmatrix} 0 & 0 & 2.5 \\ -3 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- dense matrix storage (in column-major order) treats it as a one-dimensional array:

$$[0, -3,0,0,0,0,2.5,0,0]$$

- sparse matrix storage
    - views it as a list of triplets (i, j, v) where (i, j) is a matrix coordinate and v a non-zero value
    - A would be stored as $\{(2,1,-3), (1,3,2.5)\}$

Sparse Matrices

- given an $m \times n$ matrix with k non-zero elements:
  - dense matrix storage = $8mn$ bytes
  - sparse matrix storage = $16k$ bytes
  - assuming 32-bit integers and 64-bit floating point numbers
  - sparse storage more memory-efficient as soon as $k < mn/2$
- in practice, sparse storage becomes interesting if $k \ll mn/2$, because linear algebra algorithms are vectorized

Sparse Jacobian

- the Jacobian of the deterministic problem is a sparse matrix:
    - lots of zero blocks
    - the A_s, B_s and C_s are usually also highly sparse
- family of optimized algorithms for sparse matrices (including matrix inversion for our Newton algorithm)
    - available as native objects in MATLAB/Octave (see the `sparse` command)
    - works well for medium size deterministic models
- often more efficient than Laffargue-Boucekkine-Juillard, even though it does not exploit the particular structure of the Jacobian
- default method in Dynare (`stack_solve_algo=0`)

Re-implement Algorithm in MATLAB

nk2co_understand_perfect_foresight.m