# Root Finding with Newton's Method

1. Newton's Method
   - derivation of the method
   - an implementation with SymPy and Julia
2. Convergence of Newton's Method
   - linear and quadratic convergence
   - geometric convergence for multiple roots

MCS 471 Lecture 5
Numerical Analysis
Jan Verschelde, 31 August 2022

# Root Finding with Newton's method

1. Newton's Method
   - derivation of the method
   - an implementation with SymPy and Julia
2. Convergence of Newton's Method
   - linear and quadratic convergence
   - geometric convergence for multiple roots

# derivation of Newton's method via Taylor series

Let $x_0$ be an initial approximation for a root of $f(x) = 0$.

Find $\Delta x$ so $x_1 = x_0 + \Delta x$: $f(x_1) = 0$.

Apply Taylor series to $f(x_0 + \Delta x)$:

$$f(x_0 + \Delta x) = f(x_0) + f'(x_0)\Delta x + O((\Delta x)^2).$$

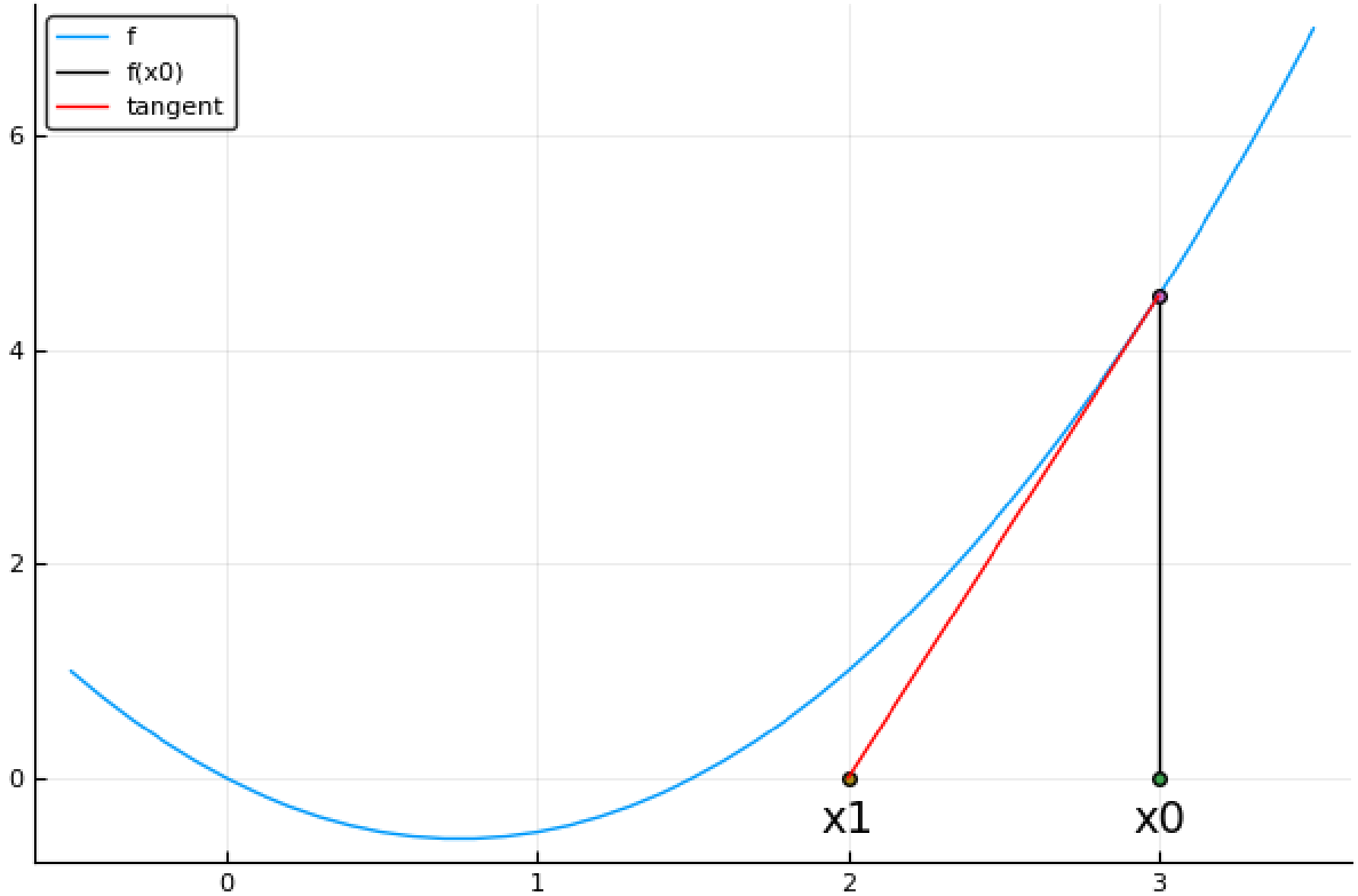
Use $f(x_1) = f(x_0 + \Delta x) = 0$ and ignore $O((\Delta x)^2)$:

$$0 = f(x_0) + f'(x_0)\Delta x.$$

Solve for $\Delta x$: $\Delta x = -\dfrac{f(x_0)}{f'(x_0)}$. Then $x_1$ is computed as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

# a geometric interpretation



# derivation of Newton's method via the tangent line

Consider the tangent line at the point $(x_0, f(x_0))$:

$$y - f(x_0) = f'(x_0)(x - x_0).$$

Set $y = 0$ and solve for $x$:

$$\begin{aligned} -f(x_0) = f'(x_0)(x - x_0) &\Leftrightarrow -\frac{f(x_0)}{f'(x_0)} = x - x_0 \\ &\Leftrightarrow x_0 - \frac{f(x_0)}{f'(x_0)} = x \end{aligned}$$

So we have the iterative formula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots$$

# an example

Newton's method:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots$$

Consider $f(x) = x^3 + x - 1 = 0$. Its derivative: $f'(x) = 3x^2 + 1$.

The iterative formula is then:

$$x_{k+1} = x_k - \frac{x_k^3 + x_k - 1}{3x_k^2 + 1} = \frac{2x_k^3 + 1}{3x_k^2 + 1}.$$

Recall the last example of fixed-point iterations in Lecture 3.

# Root Finding with Newton's method

1. Newton's Method
   - derivation of the method
   - an implementation with SymPy and Julia
2. Convergence of Newton's Method
   - linear and quadratic convergence
   - geometric convergence for multiple roots

# strings, expressions, and functions

```
julia> using SymPy

julia> x = Sym("x")
x

julia> s = "x^3 + 2*x"
"x^3 + 2*x"

julia> e = sympify(s)
 3
x  + 2·x

julia> f = lambdify(e)
#89 (generic function with 1 method)

julia> f(2)
12
```

# derivatives with SymPy

```
using SymPy
x = Sym("x")

"""
Returns the string representation of the derivative
of the expression in x, given in the string s.
"""
function SymPyDerivative(s::String)
    evaluated = sympify(s)
    derivative = diff(evaluated, x)
    return string(derivative)
end
```

Because of the first two lines, the string `s`

1. is evaluated into a SymPy expression,
2. for which `diff` applies, and
3. the derivative is then converted back into a string.

# the update function $\Delta x = -f(x)/f'(x)$

```
"""
Given a string representation of an expression in x,
returns the Julia function for the update -f(x)/f'(x),
where f(x) is the function defined by the input string.
"""
function NewtonUpdate(f::String, verbose::Bool=true)
    sdf = SymPyDerivative(f)
    sdx = string("-(", f, ")/(", sdf, ")")
    if verbose
        println("update : $sdx")
    end
    edx = sympify(sdx)
    return lambdify(edx)
end
```

A simple test:

```
dx = NewtonUpdate("x^3 + x - 1")
println("dx(1.0) : ", dx(1.0))
```

# a Julia function for Newton's method

```
"""
Runs Newton's method on the function f(x) = 0.

ON ENTRY :
    f         a function in one variable
    update    defines the update in the Newton operator
    x0        initial approximation for the root
    N         maximum number of iterations
    dxtol     tolerance on the magnitude of the update
    fxtol     tolerance on the residual

ON RETURN :
    (root, |dx|, |f(root)|, failure)
    the computed approximation for the root,
    estimates for forward and backward error,
    and a boolean to indicate failure.
"""
function Newton(f::Function, update::Function, x0::Float64,
                N::Int64=6, dxtol::Float64=1.0e-8,
                fxtol::Float64=1.0e-8)
```

# the definition of the function

```
    myroot = x0; dx = 1; y = 1.0
    stri = @sprintf("%3d", 0)
    strx = @sprintf("%.16e", myroot)
    println("step          root                        |dx|      |f(x)|")
    println("$stri  $strx")
    for i=1:N
        y = abs(Float64(f(myroot)))
        dx = Float64(update(myroot))
        myroot = myroot + dx
        stri = @sprintf("%3d", i)
        strx = @sprintf("%.16e", myroot)
        stry = @sprintf("%.2e", y)
        strdx = @sprintf("%.2e", abs(dx))
        println("$stri  $strx  $strdx  $stry")
        if((abs(dx) < dxtol) | (y < fxtol))
            return (myroot, abs(dx), y, false)
        end
    end
    return (myroot, abs(dx), y, true)
end
```

# the main function

```
"""
Prompts for an expression and an initial value
and then calls Newton's method.
"""
function main()
    println("Testing Newton's method ...")
    print("Give an expression in x : ")
    funx = readline(stdin)
    print("Give an initial value for x0 : ")
    x0line = readline(stdin)
    x0 = parse(Float64, x0line)
    f = lambdify(sympify(funx))
    update = NewtonUpdate(funx)
    result = Newton(f, update, x0)
    root = @sprintf("%.16e", result[1])
    forward = @sprintf("%.2e", result[2])
    backward = @sprintf("%.2e", result[3])
    println("root : $root ")
    println(" forward error : $forward ")
    print("backward error : $backward")
    if Bool(result[4])
        println("  Failed to converge.")
    else
        println("  Converged.")
    end
end
```

# running a test at the command prompt

```
$ julia newtonoperator.jl
Testing Newton's method ...
Give an expression in x : x^3 + x - 1
Give an initial value for x0 : 0.5
update : -(x^3 + x - 1)/(3*x^2 + 1)
step          root                    |dx|       |f(x)|
  0  5.000000000000000e-01
  1  7.142857142857143e-01  2.14e-01  3.75e-01
  2  6.831797235023041e-01  3.11e-02  7.87e-02
  3  6.823284233045782e-01  8.51e-04  2.04e-03
  4  6.823278038283471e-01  6.19e-07  1.48e-06
  5  6.823278038280194e-01  3.28e-13  7.86e-13
root : 6.823278038280194e-01
 forward error : 3.28e-13
backward error : 7.86e-13  Converged.
$
```

# two exercises

Exercise 1: Consider the sensitivity of the choice of the start value 0.5 when running Newton's method on $x^3 + x - 1$.

1. How much smaller than 0.5 can you take the initial value and still converge to the real root of $x^3 + x - 1$?
2. How much larger than the real root of $x^3 + x - 1$ can you take the initial value and still converge?

Exercise 2: The polynomial $x^3 + x - 1$ has two complex conjugated roots but Newton's method cannot compute the complex roots, unless the arithmetic is complex.
Adjust the Julia/SymPy function so it works with initial values with nonzero imaginary parts. Note that the imaginary unit in SymPy is `I`.

# Root Finding with Newton's method

1. Newton's Method
   - derivation of the method
   - an implementation with SymPy and Julia

2. Convergence of Newton's Method
   - linear and quadratic convergence
   - geometric convergence for multiple roots

# linear convergence

The bisection method converged linearly,
so did one of the fixed-point iterations of lecture 2.

If the bisection method converges, then we get one correct bit in every step, and it takes three to four steps to get one decimal place correct.

## Definition

If a sequence $x_k$ converges to $x_\infty$, denote $e_k = |x_\infty - x_k|$.
The sequence *converges linearly* if

$$\lim_{k \to \infty} \frac{e_{k+1}}{e_k} = S > 0,$$

for some positive constant $S$.

# quadratic convergence

With Newton's method we observe that the number of correct decimal places doubles in each step.

## Definition

If a sequence $x_k$ converges to $x_\infty$, denote $e_k = |x_\infty - x_k|$.
The sequence *converges quadratically* if

$$\lim_{k \to \infty} \frac{e_{k+1}}{e_k^2} = S > 0,$$

for some positive constant $S$.

# order of convergence

## Definition

The *order of convergence of a fixed-point iteration* $x_{k+1} = g(x_k)$, converging to $x_\infty$, equals $R$ if

$$g'(x_\infty) = g''(x_\infty) = \cdots = g^{(R-1)}(x_\infty) = 0$$

and $g^{(R)}(x_\infty) \neq 0$.

Applied to Newton's method, $g(x) = x - \dfrac{f(x)}{f'(x)}$.

$$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2}$$

At the fixed point $x_\infty$: $f(x_\infty) = 0$ and if $f'(x_\infty) \neq 0$: $g'(x_\infty) = 0$.

# computing square roots

For the computation of $\sqrt{7}$, consider the equation $f(x) = x^2 - 7 = 0$.

Exercise 3:

1. Construct the formula for Newton's method to derive the fixed-point formula to solve $f(x) = 0$.
2. Run Newton's method, starting at $x_0 = 7$.
   Draw a cobweb diagram.

# Root Finding with Newton's method

1. Newton's Method
   - derivation of the method
   - an implementation with SymPy and Julia

2. Convergence of Newton's Method
   - linear and quadratic convergence
   - geometric convergence for multiple roots

# multiple roots

## Definition

Consider $f(x) = 0$ with root $r$. The *multiplicity of a root* $r$ is $m$ if $f(r) = f'(r) = f''(r) = \cdots = f^{(m-1)}(r) = 0$ and $f^{(m)}(r) \neq 0$.

For a root $r$ of multiplicity $m$ of $f(x) = 0$,
the Taylor series developed at $x = r$ are

$$f(x) = \frac{f^{(m)}(r)}{m!}(x-r)^m + O((x-r)^{m+1}).$$

Denote $C = f^{(m)}(r)/m!$ and ignore higher order terms:

$$\begin{aligned} f(x) &\approx C(x-r)^m \\ f'(x) &\approx Cm(x-r)^{m-1} \\ f''(x) &\approx Cm(m-1)(x-r)^{m-2}. \end{aligned}$$

# computing the rate of convergence

For a root $r$ of multiplicity $m$ of $f(x) = 0$:

$$\begin{aligned} f(x) &\approx C(x-r)^m \\ f'(x) &\approx Cm(x-r)^{m-1} \\ f''(x) &\approx Cm(m-1)(x-r)^{m-2}. \end{aligned}$$

$$\begin{aligned} g'(x) &= 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} \\ &= 1 - \frac{C^2m^2(x-r)^{2m-2} - C(x-r)^m Cm(m-1)(x-r)^{m-2}}{C^2m^2(x-r)^{2m-2}} \\ &= 1 - \frac{m^2 - m(m-1)}{m^2} \\ &= \frac{m-1}{m} \end{aligned}$$

# geometric rate of convergence

**Theorem**

*If Newton's method converges to a root of multiplicy m, then the rate of convergence is* $\frac{m-1}{m}$.

Exercise 4: For multiplicity $m$, consider the modified Newton's method

$$x_{k+1} = x_k - m\frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots$$

Show that if the modified Newton's method converges to a root of multiplicity $m$, then the convergence is quadratic.

# another exercise

Exercise 5: Consider $f(x) = (x-1)^3$

1. Apply Newton's method to $f$ and compute the convergence rate, based on the computed errors. Do you observe $2/3$?
2. Apply the modified Newton's method with $m = 3$.
   Do you observe quadratic convergence?