Numerics – interactive

Thomas Risse
Institute for Informatics and Automation
Hochschule Bremen, Germany

December 5, 2007

Abstract

The objective of this document is to provide an oportunity to explore different basic numerical algorithms interactively.
This very document determines the precision of computation, demonstrates 2D and 3D vector calculus, solves systems of linear equations, invertes matrices, presents some numerical constants, evaluates user specified functions, exhibits the evaluation of elementary functions with their inverse functions (exponential, logarithmic, trigonometric and hyperbolic functions), introduces CORDIC algorithms, approximates zeroes, minima and integrals, and solves first order ordinary differential equations numerically.

1 Introduction

The objective of this document is to offer insight into some basic numerical algorithms by providing opportunities to run these algorithms, investigate their performance, compare different algorithms for the same purpose, solve little problems etc.

1.1 Conventions and Usage

In the following, generally:

click = , to execute an operation or evaluate a function;
click Operation , to clear arguments or results,

type input into Input -fields

read output from Output or Output -fields

The screen layout aims at presenting – in screen filling mode, i.e. window-width (Cntrl-2) for Acrobat Reader and page-width (Cntrl-3) for Acrobat Writer – all for an algorithm relevant information on one page/screen.

1.2 Precision

The *relative precision of computation*[1] ν is computed by

```
epsilon=1.0;
while (1.0+epsilon>1.0) epsilon/=2;
nu=2*epsilon;
```

For JavaScript holds JS precision $\nu =$

Ex. Precision of JavaScript computations in number of decimal digits?

1.3 Floating Point Arithmetic

Due to the limited precision of computation certain laws of arithmetic hold **no longer**, e.g. associativity $(a + b) + c = a + (b + c)$

$a =$	99999999999	$(a + b) + c =$	
$b =$	-99999999999	$a + (b + c) =$	
$c =$	0.000005		reset Add

Ex. Compute the relative error!

[1] Contrary to JavaScript or Java applets computer algebra packages like Mathematica, Maple or MuPad compute with any user specified precision.

1.4 Floating Point Arithmetic with Given Precision

The precision of JavaScript computations is quite high. To demonstrate the effects of limited precision, now the number of decimal digits in the representation of floating point numbers can be specified.

Arguments and results represented with JavaScript precision

$a =$	999999	$b =$	-999000	$c =$	0.001

$a + b =$		$(a + b) + c =$	
$b + c =$		$a + (b + c) =$	

$p = \#$ decimal digits $=$ 3 test random reset Add

Arguments and results represented by p decimal digits

$\bar{a} =$		$\bar{b} =$		$\bar{c} =$	

$\overline{(\bar{a} + \bar{b})} =$		$\overline{\overline{(\bar{a} + \bar{b})} + \bar{c}} =$	
$\overline{(\bar{b} + \bar{c})} =$		$\overline{\bar{a} + \overline{(\bar{b} + \bar{c})}} =$	

Ex. Representation of input numbers? relative error for different p ?

2 Vector Calculus

2.1 Operations on Vectors in the Plane

2.1.1 Scalar Multiples $c\,\vec{a}$ of Vectors $\vec{a}$ in the Plane

$$\boxed{} \quad \boxed{\cdot} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{=} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix}$$

2.1.2 Addition $\vec{a} + \vec{b}$ of Vectors in the Plane

$$\begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{+} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix} \boxed{=} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix}$$

2.1.3 Scalar Product $\vec{a} \cdot \vec{b}$ of Vectors in the Plane

$$\begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{\cdot} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{=} \quad \boxed{}$$

Ex.: Determine the angle $\angle(\vec{a}, \vec{b})$ between the vectors $\vec{a} = \frac{\sqrt{2}}{2}(1,1)$ and $\vec{b} = (\sqrt{3}-2, 1)$. Use section 5.

2.1.4 Length or Modulus $|\vec{a}|$ of Vectors in the Plane

$$\boxed{\text{reset}} \qquad \left|\begin{pmatrix}\boxed{}\\ \boxed{}\end{pmatrix}\right| \qquad \boxed{=} \qquad \boxed{}$$

Ex.: Normalize vectors like $\vec{a} = \frac{\sqrt{2}}{2}(1,1)$ or $\vec{b} = (\sqrt{3}-2, 1)$. Use section 5.

2.2 Operations on Vectors in Space

2.2.1 Scalar Multiples $c\vec{a}$ of Vectors $\vec{a}$ in Space

$$\boxed{} \quad \boxed{\cdot} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{=} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix}$$

2.2.2 Addition $\vec{a} + \vec{b}$ of Vectors in Space

$$\begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{+} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \boxed{=} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix}$$

2.2.3 Scalar Product $\vec{a} \cdot \vec{b}$ of Vectors in Space

$$\begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{\cdot} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{=} \quad \boxed{}$$

2.2.4 Length or Modulus $|\vec{a}|$ of Vectors in Space

$$\boxed{\text{reset}} \qquad \left|\begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix}\right| \qquad \boxed{=} \qquad \boxed{}$$

2.2.5 Vector Product $\vec{a} \times \vec{b}$ of Vectors in Space

$$\begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{\times} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix} \quad \boxed{=} \quad \begin{pmatrix}\boxed{}\\ \boxed{}\\ \boxed{}\end{pmatrix}$$

Ex.: Verify: $\vec{e}_x$, $\vec{e}_y$ and $\vec{e}_z$ are orthonormal.

Ex.: Construct a 'unit cube' with vertices in $\vec{0}$ and $\frac{\sqrt{3}}{3}(1,1,1)$.

3 Systems of Linear Equations

Systems of linear equations are given by $\mathbf{Ax} = \mathbf{b}$. For a demonstration we will use only 3×3 coefficient matrices $\mathbf{A}$, namely systems of three linear equations in three unknowns. We will compare Gauß' elimination method with and without pivotisation and the Gauß-Seidel method.

3.1 Gauß' Elimination Method

Specify coefficient matrix **A** and the vector **b** on the right hand side:

$$\mathbf{A} = \begin{pmatrix} \quad & \quad & \quad \\ \quad & \quad & \quad \\ \quad & \quad & \quad \end{pmatrix}$$

$$\mathbf{Ax} = \mathbf{A}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{A}\begin{pmatrix} \quad \\ \quad \\ \quad \end{pmatrix} = \begin{pmatrix} \quad \\ \quad \\ \quad \end{pmatrix} = \mathbf{b}$$

reset tests random save **A** and **b** det(**A**) =

eliminate x_1 then eliminate x_2

solve for x_3 solve for x_2 solve for x_1

Using the original coefficient matrix **A**, the solution **x** and the original vector **b** compute residuum $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ residuum

$$\begin{pmatrix} \quad \end{pmatrix} - \begin{pmatrix} \quad \end{pmatrix} \mathbf{x}$$

$$= \begin{pmatrix} \quad \\ \quad \\ \quad \end{pmatrix} = \mathbf{r} \text{ with } ||\mathbf{r}||_2 =$$

Ex.: Precision? Conditions for solvability? Underdetermined systems of linear equations?

3.2 Gauß' Elimination Method with Pivoting

Details to partial and complete pivoting see e.g.
`www.weblearn.hs-bremen.de/risse/MAI/docs/heath.pdf` (in german)

Specify coefficient matrix **A** and the vector **b** on the right hand side:

$$\mathbf{A} = \begin{pmatrix} \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \end{pmatrix}$$

$$\mathbf{Ax} = \mathbf{A}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{A}\begin{pmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{pmatrix} = \begin{pmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{pmatrix} = \mathbf{b}$$

reset test random save **A** and **b** $\det(\mathbf{A}) =$ solve

Using the original coefficient matrix **A**, the solution **x** and the original vector **b** compute residuum $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ residuum

$$\begin{pmatrix} \end{pmatrix} - \begin{pmatrix} \end{pmatrix}\mathbf{x}$$

$$= \begin{pmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{pmatrix} = \mathbf{r} \text{ with } ||\mathbf{r}||_2 = \boxed{}$$

Ex.: Try ill conditioned coefficient matrices. cp.
`www.weblearn.hs-bremen.de/risse/MAI/docs/heath.pdf`

3.3 Stifel's Method – SLE & Matrix Inversion

Stifel's Method solves the quadratic system of linear equations $\mathbf{Ax} = \mathbf{b}$ by solving one equation after another for one unknown after another and substituting this expression for the unknown into the other equations thereby *exchanging unknowns by constants.* Essentially, $\mathbf{Ax} = \mathbf{b}$ is solved by inverting the coefficient matrix $\mathbf{A}$ so that $\mathbf{x} = A^{-1}\mathbf{b}$.

Exchanges are performed in the following form. The so called *cellar row* holds intermediate results to be reused.

$$
\begin{array}{r|cccccc}
 & x_1 & x_2 & \dots & x_j & \dots & x_n \\
\hline
b_1 & a_{11} & a_{12} & \dots & \underline{a_{1j}} & \dots & a_{1n} \\
\vdots & & & & \vdots & & \\
b_i & \underline{a_{i1}} & \underline{a_{i2}} & \dots & \underline{\underline{a_{ij}}} & \dots & \underline{a_{in}} \\
\vdots & & & & \vdots & & \\
b_{n-1} & a_{n-1,1} & a_{n-1,2} & \dots & \underline{a_{n-1,j}} & \dots & a_{n-1,n} \\
b_n & a_{n1} & a_{n2} & \dots & \underline{a_{nj}} & \dots & a_{nn} \\
\hline
\text{cellar} & & & & \text{—} & & \\
\end{array}
$$

b_i against x_j is exchanged by the following procedure:

```
prepare  pivot $= a_{ij}$; for ($k \neq j$) cellar[k] = $-a_{ik}$/pivot;

exchange $a_{ij}$ = 1/pivot; for ($k \neq i$) $a_{kj}$ /= pivot;
    for ($k \neq j$) $a_{kj}$=tmp[k];
    for ($u \neq i$) for ($v \neq j$) $a_{uv}$ += $a_{uj}$*tmp[v];
```

Ex.: Check the invariance of *column vector on the left of the form = Matrix times transposed row vector top of form.*

reset test random save **A**

	x_1	x_2	x_3
b_1			
b_2			
b_3			
cellar			

with det(**A**) =

1st click:	fill cellar
2nd click:	set other elements
3rd click:	set pivot column
4th click:	set pivot row

1 ↲ 1	1 ↲ 2	1 ↲ 3
2 ↲ 1	2 ↲ 2	2 ↲ 3
3 ↲ 1	3 ↲ 2	3 ↲ 3

verify $\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{I}$

$$\mathbf{A}\,\mathbf{A}^{-1} = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

$$\begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

$$\approx \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

Ex.: Precision? Impact of different order of exchanges?

3.4 Gauß-Seidel Method

Assuming that all main diagonal elements of the coeffizient matrix are not zero, i.e. $a_{ii} \neq 0$ for $i = 1, \ldots, n$, then the equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be solved for the unknowns on the main diagonal, e.g. for $n = 3$

$$\begin{aligned} x_1 &= \tfrac{1}{a_{11}} \left(b_1 - a_{12}x_2 - a_{13}x_3\right) \\ x_2 &= \tfrac{1}{a_{22}} \left(b_2 - a_{21}x_1 - a_{23}x_3\right) \\ x_3 &= \tfrac{1}{a_{33}} \left(b_3 - a_{31}x_1 - a_{32}x_2\right) \end{aligned}$$

Using

$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{a_{11}} \left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}\right) \\ x_2^{(k+1)} &= \tfrac{1}{a_{22}} \left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)}\right) \\ x_3^{(k+1)} &= \tfrac{1}{a_{33}} \left(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)}\right) \end{aligned}$$

and specifying a start vector $\mathbf{x}^{(0)}$, a solution $\mathbf{x}^{(k)}$ is inserted and improved on the basis of already improved components. In this way $\mathbf{x}^{(k)}$ is transformed into $\mathbf{x}^{(k+1)}$.

The method converges if for example – after reordering – the main diagonal elements dominate the other elements in the corresponding row, i.e. $|a_{i,i}| > \sum_{j=1,j\neq i}^{n} |a_{i,j}|$ for $i = 1, \ldots, n$.

Specify coeffizient matrix $\mathbf{A}$ and vector $\mathbf{b}$ on the right hand side:

$$\mathbf{A} = \begin{pmatrix} \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \end{pmatrix}$$

$$\mathbf{A}\mathbf{x}^{(k+1)} = \begin{pmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{pmatrix} \approx \begin{pmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \end{pmatrix} = \mathbf{b}$$

start vector $\mathbf{x}^{(0)} = \begin{pmatrix} \boxed{0} \\ \boxed{0} \\ \boxed{0} \end{pmatrix}$, $\det(\mathbf{A}) =$

$\boxed{\text{reset}}$ $\boxed{\text{test}}$ $\boxed{\text{random } \mathbf{A}}$ $\boxed{\text{random } \mathbf{b}}$ $\boxed{\text{random } \mathbf{x}^{(0)}}$ $\boxed{\text{convergence?}}$

$k =$ $\qquad\qquad\qquad$ $k+1 =$

$$\mathbf{x}^{(k)} = \begin{pmatrix} \boxed{0} \\ \boxed{0} \\ \boxed{0} \end{pmatrix} \qquad \mathbf{x}^{(k+1)} = \begin{pmatrix} \boxed{0} \\ \boxed{0} \\ \boxed{0} \end{pmatrix}$$

$\boxed{x_1^{(k)} \to x_1^{(k+1)}}$ $\boxed{x_2^{(k)} \to x_2^{(k+1)}}$ $\boxed{x_3^{(k)} \to x_3^{(k+1)}}$ oder zusammen $\boxed{k \to k+1}$

Ex.: Dependance on the start vector? Convergence speed?

Ex.: Compare the direct (Gauß) and the iterative (Gauß-Seidel) method!

4 Some Constants

$e =$		$\pi =$	
$\ln 2 =$		$\ln 10 =$	
$\mathrm{ld}\ e = \log_2 e =$		$\lg\ e = \log_{10} e =$	
$\sqrt{1/2} =$		$\sqrt{2} =$	

get constants

Ex.: Compare the built in constants with the computed values of suitable functions in section 5.

5 Evaluation of Functions

Any function can be evaluated which is specified by an algebraic expression consisting of the elementary functions `abs`, `chi`, `pow`, `sqrt`, `exp`, `ln`, `sin`, `cos`, `tan`, `cot`, `arcsin`, `arccos`, `arctan`, `arccot`, `sinh`, `cosh`, `tanh`, `coth`, `arsinh`, `arcosh`, `artanh`, `arcoth`, `factorial`, `gamma`, `loggamma` and the constants `e` and `pi` (written in exactly this way) – functions of the one independent variable `x`. The characteristic function `chi(x,a,b)` is defined by $\chi_{[a,b]}(x) = \begin{cases} 1 & x \in [a,b] \\ 0 & \text{sonst} \end{cases}$ with $a < b$. Any **constant** expression using the above mentioned functions and constants is allowed for the argument `x` also.

$f(x) =$

i.e. $f(x) =$

f( )= test

Ex.: Compute $\sin(x)$ and $\cos(x)$ for $x = 30^o, 45^o, 60^o, \ldots$ specifying and evaluating suitable functions sin and cos.

Ex.: Evaluate in degrees $\arctan(x)$ for $x = 1, \sqrt{3}, \ldots$.

Ex.: Evaluate functions like $\operatorname{arsinh}(x)$ or $\operatorname{arcosh}(x)$ for $x = 0, 1, \ldots$.

Ex.: What happens to $\frac{\sin(x)}{x}$ for $0 < x \ll 1$ when determining $\lim_{x\to 0} \frac{\sin(x)}{x}$?

Ex.: Compare `gamma(n)` and `factorial(n-1)`.

6 Elementary Functions with Inverse

6.1 Exponential Function and Logarithm

6.1.1 Exponential Function

$\exp(\qquad) = \qquad$

$\qquad = \ln(\qquad)$

Graph

6.1.2 Logarithm

$\ln(\qquad) = \qquad$

$\qquad = \exp(\qquad)$

Graph

Ex.: Compute $e^{\ln c}$ for $c > 0$ and $\ln e^{d}$ for any real d.

6.2 Trigonometric Functions with their Inverse Functions

6.2.1 Sine and Arc Sine

sin(arcus degree) =

= arcsin( )

Graph

6.2.2 Cosine and Arc Cosine

cos(arcus degree)=

= arccos( )

Graph

6.2.3 Tangent and Arc Tangent

tan(arcus degree)=

= arctan( )

Graph

6.2.4 Cotangent and Arc Cotangent

cot(arcus degree)=

= arccot( )

Graph

Ex.: Werte für $0, \pi/6, \pi/4, \pi/3$ und $\pi/2$, Periodizizät, Symmetrie

Ex.: $\sin x \approx x$, $\cos x \approx 1$, $\tan x \approx x$ für $|x| \ll 1$,

6.3 Hyperbolic Functions with their Inverse Functions

6.3.1 Hyperbolic Sine

sinh( ) =

=arsinh( )

Graph

6.3.2 Hyperbolic Cosine

cosh( ) =

=arcosh( )

Graph

6.3.3 Hyperbolic Tangent

tanh( ) =

=artanh( )

Graph

6.3.4 Hyperbolic Cotangent

coth( ) =

=arcoth( )

Graph

Ex.: Determine asymptotic lines of tanh and coth! Which problems are encountered for artanh(tanh(x)) and for arcoth(coth(x)) resp.?

Ex.: Implementations of sinh, cosh, tanh and coth are straight forward. *textbook/online search:* How to implement the inverse functions arsinh, arcosh, artanh and arcoth if only the logarithm `ln` and the square root `sqrt` are available?

7 CORDIC – Evaluation of Elementary Functions in Hardware

CORDIC is the acronym of ¨*COordinate Rotation DIgital Computer*¨. The CORDIC algorithms compute some elementary functions by nearly only fast operations, namely additions (*adds*) and multiplications by powers of two (*shifts*).
Therefore, CORDIC algorithms are very well suited for implementations in (fixed point) hardware.

Rotation of vectors in the (complex) plane by the angle φ and the origin as fix point it given by

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\varphi - y\sin\varphi \\ x\sin\varphi + y\cos\varphi \end{pmatrix} = \cos\varphi \begin{pmatrix} x - y\tan\varphi \\ y + x\tan\varphi \end{pmatrix}$$

Especially for $\varphi = \pm\arctan(2^{-i})$ and hence for $\tan\varphi = \pm 2^{-i}$ holds

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \cos\varphi \begin{pmatrix} x \mp 2^{-i}\, y \\ y \pm 2^{-i}\, x \end{pmatrix} = \frac{1}{\sqrt{1+2^{-2i}}} \begin{pmatrix} x \mp 2^{-i}\, y \\ y \pm 2^{-i}\, x \end{pmatrix}$$

Except for the multiplication by the scalar $\frac{1}{\sqrt{1+2^{-2i}}}$ the rotation by $\arctan 2^{-i}$ can be computed by *adds* and *shifts* alone. Sequencing such rotations gives

$$\begin{aligned}
\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} &= \frac{1}{\sqrt{1+2^{-2i}}} \begin{pmatrix} x_i \mp 2^{-i}\, y_i \\ y_i \pm 2^{-i}\, x_i \end{pmatrix} = \frac{1}{\sqrt{1+2^{-2i}}} \begin{pmatrix} 1 & \mp 2^{-i} \\ \pm 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \\
&= \tfrac{1}{\sqrt{1+2^{-2i}}} \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} \tfrac{1}{\sqrt{1+2^{-2(i-1)}}} \begin{pmatrix} 1 & -\delta_{i-1} 2^{-(i-1)} \\ \delta_{i-1} 2^{-(i-1)} & 1 \end{pmatrix} \\
&\qquad \cdots \quad \tfrac{1}{\sqrt{1+2^{-2}}} \begin{pmatrix} 1 & -\delta_1 2^{-1} \\ \delta_1 2^{-1} & 1 \end{pmatrix} \tfrac{1}{\sqrt{1+2^{0}}} \begin{pmatrix} 1 & -\delta_o 2^{0} \\ \delta_o 2^{0} & 1 \end{pmatrix} \begin{pmatrix} x_o \\ y_o \end{pmatrix} \\
&= \textstyle\prod_{i=0}^{n} \tfrac{1}{\sqrt{1+2^{-2i}}} \displaystyle\prod_{i=0}^{n} \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_o \\ y_o \end{pmatrix} = g_n \prod_{i=0}^{n} \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_o \\ y_o \end{pmatrix}
\end{aligned}$$

where $\delta_i = \pm 1$ indicates the direction of the i^{th} rotation and the algorithms gain g_∞ is

$$g_\infty = \prod_{i=0}^{\infty} \frac{1}{\sqrt{1+2^{-2i}}} = \lim_{n\to\infty} g_n = \lim_{n\to\infty} \prod_{i=0}^{n} \frac{1}{\sqrt{1+2^{-2i}}} \approx 1.6467602581210654$$

Ex.: Formulate the results above using the addition theorems for sine and cosine alone.

7.1 Trigonometric Functions (circular $m = 1$, rotating)

The vectors $\vec{z}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ with $z_o = \vec{e}_x$ approximate the vector $g \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix}$, if the rotation by φ is approximated by a sequence of rotations by $\pm \arctan 2^{-i}$. The following algorithm approximates $\sin\varphi$ and $\cos\varphi$ for φ given in radians with $|\varphi| < \frac{\pi}{2}$. `arctan(k)` is to be implemented by a table look up.

```
// return (cos(phi), sin(phi))
g=1.6467602581210654; k=1; // k= 2^{-i}
x=1; y=0;                  // e_x = (x,y)
do {
   kk=k; if (phi<0) kk=-k;
   tmpx=x-kk*y; tmpy=y+kk*x;
   x=tmpx; y=tmpy; phi-=arctan(kk); k/=2;
} while (abs(phi)>epsilon);
return (x/g,y/g); // return (cos φ, sin φ)
```

Ex.: Design the look up table for `atan(kk)` !

Ex.: What systematic error has to be tolerated if one wants to avoid multiplications except for the last two divisions by g ?

`Math.sin`, `Math.cos` and `Math.tan`, the library functions provided by JavaScript, serve as references (in green). Input 'symbolic' $\varphi \in [-\frac{\pi}{2}, \frac{\pi}{2}] \approx [-1.57, 1.57]$, e.g. `pi/5`, `0.5` or `arcsin(0.5)`.

$\varepsilon =$	0.0000001	get φ and evaluate library functions	
symbolic $\varphi =$		$z = \varphi =$	
$n =$	0	$x_n =$	1
$0 \leftarrow z_n = \varphi_n =$	z	$y_n =$	0
$\text{gain}_n =$	1	test step	cont reset
$\text{gain}_\infty =$	1.6467602581210654		compute gain_∞
$x_n/\text{gain}_\infty =$		$y_n/\text{gain}_\infty =$	
`Math.cos`$(\varphi) =$		`Math.sin`$(\varphi) =$	
$y_n/x_n =$		$x_n/y_n =$	
`Math.tan`$(\varphi) =$		`1/Math.tan`$(\varphi) =$	

Ex.: Which precision is gained in each step of the CORDIC algorithm?

Ex.: What does happen when applying this version of the CORDIC-algorithm to arguments like $\frac{\pi}{4}$? Which are the other critical arguments?

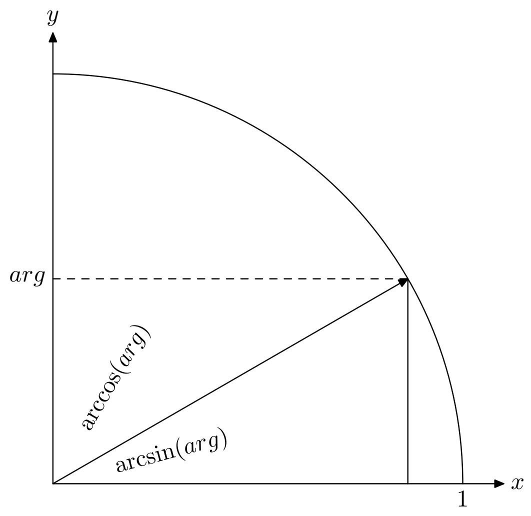
7.1.1 Transforming Polar to Cartesian Coordinates

By the simultaneous computation of sine and cosine, the polar-coordinates (r, φ) of a vector can be transformed to the Cartesian coordinates $(x, y) = r(\cos\varphi, \sin\varphi)$ by rotating the Cartesian start vector $(r, 0)$ to become the target vector $r(\cos\varphi, \sin\varphi)$.

7.2 Inverse Trigonometric Functions (circular $m = 1$, vectoring)

In order to compute $\varphi = \arcsin(arg)$, the unit vector $\vec{e}_x$ is rotated until the y-coordinate of the rotated vector becomes arg. Then $\arccos(arg)$ can be computed by

$$\arccos(arg) = \tfrac{\pi}{2} - \arcsin(arg)$$



`Math.asin`, `Math.acos` and `Math.atan` are library functions provided by JavaScript. Of course, valid 'symbolic' arguments a of arcus sine and arcus cosine are in $[-1, 1]$, e.g. $a =$`sin`(0.5) or $a =$`sqrt`$(3)/2$.

$\varepsilon =$	0.0000001		get arg and evaluate library functions
symbolic $a =$		$a =$	

test	step	cont	reset

$\text{gain}_\infty =$	1.6467602581210654	compute gain_∞

$n =$	0	$z_n =$	0
$\text{gain}_n =$	1	`Math.asin`$(a) =$	
$x_n =$	0.6072529350088813	$\frac{\pi}{2} - z_n =$	1.5707963267948966
$a \leftarrow y_n =$	0	$\frac{\pi}{2}-$`Math.asin`$(a) =$	

$n =$	0	$z_n =$	0
$\text{gain}_n =$	1	`Math.atan`$(a) =$	
$x_n =$	1	$\frac{\pi}{2} - z_n =$	1.5707963267948966
$0 \leftarrow y_n =$	a	$\frac{\pi}{2}-$`Math.atan`$(a) =$	

7.2.1 Transforming Cartesian to Polar Coordinates

In order to transform Cartesian coordinates (x, y) to polar coordinates (r, φ) both $r = \sqrt{x^2 + y^2}$ and $\varphi = \arctan \frac{y}{x}$ are to be computed: the CORDIC algorithm to compute the arctangent produces both when it totates the start vector (x, y) to become the Cartesian vector $(r, 0)$. Because length is preserved this implies $r = \sqrt{x^2 + y^2}$. The rotation angle is $\varphi = \arctan \frac{y}{x}$.

7.3 Hyperbolic Functions (hyperbolic $m = -1$, rotating)

Rotation of the (complex) plane is based on the addition theorems of sine and cosine. For the hyperbolic functions sinh and cosh there hold the following (corresponding) addition theorems

$$\begin{aligned}\cosh(x \pm y) &= \cosh(x)\cosh(y) \pm \sinh(x)\sinh(y)\\ \sinh(x \pm y) &= \sinh(x)\cosh(y) \pm \cosh(x)\sinh(y)\end{aligned}$$

These theorems allow to evaluate $\cosh a$ and $\sinh a$ for a given argument a by a sequence of 'hyperbolic' rotations

$$\begin{pmatrix}\cosh(x \pm y)\\ \sinh(x \pm y)\end{pmatrix} = \begin{pmatrix}\cosh y & \pm\sinh y\\ \pm\sinh y & \cosh y\end{pmatrix}\begin{pmatrix}\cosh x\\ \sinh x\end{pmatrix} = \cosh y \begin{pmatrix}1 & \pm\tanh y\\ \pm\tanh y & 1\end{pmatrix}\begin{pmatrix}\cosh x\\ \sinh x\end{pmatrix}$$

using suitable $y = \operatorname{artanh} 2^{-i}$ or $\tanh y = 2^{-i}$ these rotations can be performed (except for the scaling at the end) by shifts and adds only.

$$\begin{aligned}\begin{pmatrix}\cosh(x \pm y)\\ \sinh(x \pm y)\end{pmatrix} &= \cosh \operatorname{artanh} 2^{-i}\begin{pmatrix}1 & \pm 2^{-i}\\ \pm 2^{-i} & 1\end{pmatrix}\begin{pmatrix}\cosh x\\ \sinh x\end{pmatrix}\\ &= \frac{1}{\sqrt{1-2^{-2i}}}\begin{pmatrix}1 & \pm 2^{-i}\\ \pm 2^{-i} & 1\end{pmatrix}\begin{pmatrix}\cosh x\\ \sinh x\end{pmatrix}\end{aligned}$$

where $\frac{1}{\cosh y} = \sqrt{\frac{\cosh^2 y - \sinh^2 y}{\cosh^2 y}} = \sqrt{1 - \tanh^2 y}$ für $y = \pm \operatorname{artanh} 2^{-i}$ implies $\cosh(\pm \operatorname{artanh} 2^{-i}) = \frac{1}{\sqrt{1-2^{-2i}}}$.

To evaluate $\cosh a$ and $\sinh a$ at the same time, a has to be represented as sum $a = \sum d_i \operatorname{artanh} 2^{-i}$ of suitable 'rotation angles' $\pm \operatorname{artanh} 2^{-i}$ and 'rotation directions' $d_i \in \{1, -1\}$. Starting with $\cosh 0 = 1$ and $\sinh 0 = 0$, $\cosh a$ and $\sinh a$ are computed as sequence of such 'rotations'

$$\begin{pmatrix}\cosh a\\ \sinh a\end{pmatrix} \leftarrow \prod_{i=0}^{n} \frac{1}{\sqrt{1-2^{-2i}}}\begin{pmatrix}1 & d_i 2^{-i}\\ d_i 2^{-i} & 1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = g_n \prod_{i=0}^{n}\begin{pmatrix}1 & d_i 2^{-i}\\ d_i 2^{-i} & 1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix}$$

The *gain* g_∞ of the algorithmus is

$$g_\infty = \prod_{i=0}^{\infty} \frac{1}{\sqrt{1-2^{-2i}}} = \lim_{n\to\infty} g_n = \lim_{n\to\infty} \prod_{i=0}^{n} \frac{1}{\sqrt{1-2^{-2i}}} \approx$$

```
g=0.6; k=1; // k= $2^{-i}$
x=1; y=0;   // $\vec{e}_x =$ (x,y)
do {
   kk=k; if (phi<0) kk=-k;
   tmpx=x+kk*y; tmpy=y+kk*x;
   x=tmpx; y=tmpy; arg-=artanh(kk); k/=2;
} while (abs(arg)>epsilon);
return (x/g,y/g); // return $(\cosh(arg), \sinh(arg))$
```

Ex.: Design the look up table for `artanh(kk)` !

Ex.: What systematic error has to be tolerated if one wants to avoid multiplications except for the last two divisions by g ?

The locally provided functions `cosh`, `sinh`, `tanh` and `coth` as well as the JavaScript library function `Math.exp` serve as reference (in green).

$\varepsilon =$	0.0000001	get a and evaluate library functions	
symbolic $a =$		$z = a =$	
$n =$	1	$\begin{pmatrix} x_n \\ y_n \end{pmatrix} =$	1
$0 \leftarrow z_n = a_n =$	z		0
$\text{gain}_n =$	1	test step	cont reset
$\text{gain}_\infty =$	1.2577751496401886		compute gain_∞
$x_n{*}\text{gain}_\infty =$		$y_n{*}\text{gain}_\infty =$	
`cosh`$(a) =$		`sinh`$(a) =$	
$y_n/x_n =$		$x_n/y_n =$	
`tanh`$(a) =$		`coth`$(a) =$	
$(x_n{+}y_n)\text{gain}_\infty =$		$\varepsilon_1 + 2\sum_{i=2}^{\infty} \varepsilon_i$	$\approx$
`Math.exp`$(a) =$		artanh ( )	$=$

Ex.: Which arguments to the above algorithm are valid?

Ex.: How to deal with non valid arguments? Represent any argument $x \in \mathbb{R}$ as sum of an admissible argument z with $|z| \leq 1$ and a multiple of $\ln 2$ and consider the addition theorems. With $x = z + m \ln 2$

$$\cosh x = \cosh(z + m \ln) = \cosh z \cosh(m \ln 2) + \sinh z \sinh(m \ln 2)$$

$$\sinh x = \sinh(z + m \ln) = \sinh z \cosh(m \ln 2) + \cosh z \sinh(m \ln 2)$$

where $\cosh(m \ln 2) = \frac{1}{2}(2^m + 2^{-m})$ and $\sinh(m \ln 2) = \frac{1}{2}(2^m - 2^{-m})$.

7.3.1 Exponential Function and Other Hyperbolic Functions (hyperbolic $m = -1$, rotating)

As in the case of the trigonometric functions, and due to

$$\tanh x = \tfrac{\sinh x}{\cosh x} \quad \text{and} \quad \coth x = \tfrac{\cosh x}{\sinh x} \quad \text{and} \quad \exp x = \sinh x + \cosh x$$

the hyperbolic functions tanh and coth as well as the exponential function are easily computed by CORDIC algorithms.

7.4 Invers Hyperbolic Functions (hyperbolic $m = -1$, vectoring)

CORDIC algorithms compute the inverse hyperbolic functions in the same way as the trigonometric functions and their inverses.

The locally provided function `artanh` and the JavaScript library function `Math.sqrt` serve as reference (in green).

$\varepsilon =$	0.0000001		get a and evaluate library functions
symbolic $y_1 =$		$y_1 =$	
symbolic $x_1 =$		$x_1 =$	
$n =$	1	$\begin{pmatrix} x_\infty \\ 0 \end{pmatrix} \leftarrow \begin{pmatrix} x_n \\ y_n \end{pmatrix} =$	x1
$y_1/x_1 =$			y1
$\text{gain}_n =$	1	test step cont	reset
$\text{gain}_\infty =$	1.2577751496401886		compute gain_∞
$z_n =$	0	$x_n * \text{gain}_\infty =$	
$\texttt{artanh}(\frac{y_1}{x_1}) =$		$\sqrt{x_1^2 - y_1^2} =$	

Because of $\text{arcoth}(x) = \text{artanh}(\frac{1}{x})$ besides

$$\text{arsinh}(x) = \text{artanh}(\tfrac{x}{\sqrt{x^2+1}}) \quad \textit{and} \quad \text{arcosh}(x) = \text{artanh}(\tfrac{\sqrt{x^2-1}}{x})$$

the CORDIC algorithm for artanh computes the other inverse hyperbolic functions as well.

Ex.: Wich arguments in the algorithm above are not valid? How to deal with these non valid arguments?

7.4.1 Logarithm and Square Root (hyperbolic $m = -1$, vectoring)

Because $\tanh \ln \sqrt{x} = \frac{e^{\ln \sqrt{x}} - e^{-\ln \sqrt{x}}}{e^{\ln \sqrt{x}} + e^{-\ln \sqrt{x}}} = \frac{\sqrt{x} - 1/\sqrt{x}}{\sqrt{x} + 1/\sqrt{x}} = \frac{x-1}{x+1}$ and $\ln x = 2 \,\text{artanh}\, \frac{x-1}{x+1}$ there is a CORDIC algorithm to evaluate the logarithm.

To compute $\sqrt{r} = \sqrt{(r+\frac{1}{4})^2 - (r-\frac{1}{4})^2}$ let $x_o = r + \frac{1}{4}$, $y_o = r - \frac{1}{4}$ and

$\alpha = \operatorname{artanh}\frac{r-\frac{1}{4}}{r+\frac{1}{4}} = \operatorname{artanh}\frac{y_o}{x_o}$ so that

$$\begin{aligned} x_o\cosh\alpha - y_o\sinh\alpha &= x_o\cosh\operatorname{artanh}\frac{y_o}{x_o} - y_o\sinh\operatorname{artanh}\frac{y_o}{x_o} \\ &= x_o\frac{1}{\sqrt{1-\frac{y_o^2}{x_o^2}}} - y_o\frac{\frac{y_o}{x_o}}{\sqrt{1-\frac{y_o^2}{x_o^2}}} \\ &= \frac{x_o^2}{\sqrt{x_o^2-y_o^2}} - \frac{y_o^2}{\sqrt{x_o^2-y_o^2}} = \sqrt{x_o^2-y_o^2} \end{aligned}$$

7.5 Multiplication (linear $m = 0$, rotating)

There is a CORDIC-version of (fixed or floating point) multiplication.

$\epsilon =$	0.0000001	$x_o =$			$z_o =$		
$n =$	0	$\begin{pmatrix} x_n \\ y_n \end{pmatrix} =$	x0			$\rightarrow \begin{pmatrix} x_o \\ x_o\, z_o \end{pmatrix}$	
$0 \leftarrow z_n =$	z0		0				
$x_o\, z_o =$		test	step		cont		reset

Ex.: How to cope with the limited argument domain?

7.6 Division (linear $m = 0$, vectoring)

There is a CORDIC-version of (fixed or floating point) division.

$\epsilon =$	0.00001	$y_o =$			$x_o =$		
$n =$	0	$\begin{pmatrix} x_n \\ y_n \end{pmatrix} =$	x0			$\rightarrow \begin{pmatrix} x_o \\ 0 \end{pmatrix}$	
$z_n =$	0		y0				
$y_o/x_o =$		test	step		cont		reset

Ex.: How to cope with the limited argument domain?

7.7 CORDIC-Algorithms unified

CORDIC-algorithms of type *rotating* and *vectoring* in the three modi linear ($m = 0$), circular ($m = 1$) and hyperbolic ($m = -1$) can be represented in a common way and thus together can be implemented efficiently. Let T_k be the matrix

$$T_k^{m=0} = \begin{pmatrix} 1 & 0 \\ \delta_k\, 2^{-k} & 1 \end{pmatrix} \quad T_k^{m=1} = \begin{pmatrix} 1 & -\delta_k\, 2^{-k} \\ \delta_k\, 2^{-k} & 1 \end{pmatrix} \quad T_k^{m=-1} = \begin{pmatrix} 1 & -\delta_k\, 2^{-k} \\ \delta_k\, 2^{-k} & 1 \end{pmatrix}$$

Then, the following table gives the details.

mode m	rotating $z_n \to 0$ $z_{k+1} = z_k - \delta_k \epsilon_k$ $\delta_k = \operatorname{sgn} z_k$	vectoring $y_n \to 0$ $y_{k+1} = y_k ? \delta_k \epsilon_k$ $\delta_k = -\operatorname{sgn} y_k$
$m = 0$ $\epsilon_k = 2^{-k}$ $g = 1$	multiplication $x_o\, z_o$ $\prod_{k=0}^{n} T_k \begin{pmatrix} x_o \\ 0 \end{pmatrix} \to \begin{pmatrix} x_o \\ x_o\, z_o \end{pmatrix}$	division, $z_o = 0$ $\prod_{k=0}^{n} T_k \begin{pmatrix} x_o \\ y_o \end{pmatrix} \to \begin{pmatrix} x_o \\ 0 \end{pmatrix}$ where $z_k \to y_o/x_o$
$m = 1$ $\epsilon_k = \arctan 2^{-k}$ $g = \prod_{k=0}^{n} \cos \epsilon_k$	trigonometric $\prod_{k=0}^{n} T_k \begin{pmatrix} g \\ 0 \end{pmatrix} \to \begin{pmatrix} \cos z_o \\ \sin z_o \end{pmatrix}$	inverse trigonometric, $z_o = 0$ $\prod_{k=0}^{n} T_k \begin{pmatrix} x_o \\ y_o \end{pmatrix} \to \frac{1}{g} \begin{pmatrix} \sqrt{x_o^2 + y_o^2} \\ 0 \end{pmatrix}$ where $z_k \to \arctan \frac{y_o}{x_o}$
$m = -1$ $\epsilon_k = \operatorname{artanh} 2^{-k}$ $g = \prod_{k=0}^{n} \cosh \epsilon_k$	hyperbolic $\prod_{k=0}^{n} T_k \begin{pmatrix} g \\ 0 \end{pmatrix} \to \begin{pmatrix} \cosh z_o \\ \sinh z_o \end{pmatrix}$	inverse hyperbolic, $z_o = 0$ $\prod_{k=0}^{n} T_k \begin{pmatrix} x_o \\ y_o \end{pmatrix} \to \frac{1}{g} \begin{pmatrix} \sqrt{x_o^2 - y_o^2} \\ 0 \end{pmatrix}$ where $z_k \to \operatorname{artanh} \frac{y_o}{x_o}$

7.8 Convergence of CORDIC Algorithms

Convergence of CORDIC Algorithms is based on the following theorem.

Theorem
Let $e_o \geq e_1 \geq e_2 \geq \ldots \geq e_n > 0$ be a finite sequence of real numbers with

$$e_k \leq e_n + \sum_{j=k+1}^{n} e_j \quad \text{for all} \quad 0 \leq k \leq n.$$

Let r be some real number with $|r| \leq \sum_{j=0}^{n} e_j$. Let the sequence s_k be inductively defined by $s_o = 0$ and $s_{k+1} = s_k + e_k \operatorname{sgn}(r - s_k)$ for $k \in \mathbb{N}_o$. Then

$$|r - s_k| \leq e_n + \sum_{j=k}^{n} e_j \quad \text{and in particular} \quad |r - s_{n+1}| \leq e_n. \quad \text{i.e.} \quad \lim_{k\to\infty} s_k = r$$

Proof by induction:
assume $|r - s_k| \leq e_n + \sum_{j=k}^{n} e_j$. To show $|r - s_{k+1}| \leq e_n + \sum_{j=k+1}^{n} e_j$ consider the four cases:

$\boldsymbol{r > s_k}$ Then $|r - s_k| = r - s_k \leq e_n + \sum_{j=k}^{n} e_j$ and $\operatorname{sgn}(r - s_k) = 1$.

- $\boldsymbol{r - s_k - e_k > 0}$ Then $|r - s_{k+1}| = r - s_k - e_k \leq e_n + \sum_{j=k}^{n} e_j - e_k = e_n + \sum_{j=k+1}^{n} e_j$.
- $\boldsymbol{r - s_k - e_k < 0}$ i.e. $s_k < r < s_k + e_k$ or $0 < r - s_k < e_k$ or $-e_k < s_k - r < 0$ so that $|r - s_{k+1}| = |r - s_k - e_k| = s_k - r + e_k \leq e_k \leq e_n + \sum_{j=k+1}^{n} e_j$.

$\boldsymbol{r < s_k}$ Then $|r - s_k| = s_k - r \leq e_n + \sum_{j=k}^{n} e_j$ and $\operatorname{sgn}(r - s_k) = -1$.

- $\boldsymbol{r - s_k + e_k > 0}$ Then $|r - s_{k+1}| = r - s_k + e_k \leq e_k \leq e_n + \sum_{j=k+1}^{n} e_j$.
- $\boldsymbol{r - s_k + e_k < 0}$ Then $|r - s_{k+1}| = |r - s_k + e_k| = s_k - r - e_k \leq e_n + \sum_{j=k}^{n} e_j - e_k = e_n + \sum_{j=k+1}^{n} e_j$.

qed

Corollary
The e_k of the CORDIC algorithms în the three modi satisfy the conditions of the above theorem so that the CORDIC algorithms are convergent.

Proof

$\boldsymbol{m = 0}$ Then $e_k = 2^{-k}$ with $e_o \geq e_1 \geq \ldots \geq e_n > 0$ and $e_k = 2^{-k} \leq e_n + \sum_{j=k+1}^n e_j$ because for the right hand side holds $e_n + \sum_{j=k+1}^n e_j = 2^{-n} + \sum_{j=k+1}^n 2^{-j} = 2^{-n} + 2^{-(k+1)} \sum_{i=0}^{n-k-1} 2^{-i} = 2^{-n} + 2^{-(k+1)} \frac{1-2^{-(n-k)}}{1-2^{-1}} = 2^{-n} + 2^{-(k+1)}\, 2(1 - 2^{-(n-k)}) = 2^{-n} + 2^{-k}(1 - 2^{-(n-k)}) = 2^{-k} = e_k$.

$\boldsymbol{m = 1}$ Then $e_k = \arctan 2^{-k}$ with $e_o \geq e_1 \geq \ldots \geq e_n > 0$ because of the monotony of Arcus Tangens. Due to $2 \arctan x = \arctan \frac{2\,x}{1-x^2}$ and the monotony $\arctan x \leq 2 \arctan \frac{x}{2} = \arctan \frac{2\,x/2}{1-x^2/4}$, i.e.

$$e_k \leq 2\, e_{k+1}$$

and by repetition

$$e_k \leq 2\, e_{k+1} \leq e_{k+1} + 2\, e_{k+2} \leq \ldots \leq e_n + \sum_{j=k+1}^{n} e_j$$

$\boldsymbol{m = -1}$ Then $e_k = \operatorname{artanh} 2^{-k} = \frac{1}{2} \ln \frac{1+2^{-k}}{1-2^{-k}}$. To guarantee convergence, all hyperbolic rotations are duplicated except for the first. Hence

$$e_k \leq e_n + 2 \sum_{j=k+1}^{n} e_j \quad \text{for} \quad k = 1, 2, \ldots, n$$

is to be proven, namely by induction in $n - k$, i.e.

$$e_{n-k} \leq e_n + 2 \sum_{j=n-k+1}^{n} e_j \quad \text{for} \quad k = 1, 2, \ldots, n-1$$

In case $k = 1$ then $e_{n-1} \leq e_n + 2\, e_n = 3\, e_n$ to be shown which is eqivalent to

$$0 \leq 3\, e_n - e_{n-1} = \tfrac{1}{2} \ln \Big(\big(\tfrac{1+2^{-n}}{1-2^{-n}}\big)^3 \tfrac{1-2^{-n+1}}{1+2^{-n+1}} \Big)$$

which due to monotony of the logarithm is eqivalent to

$$\big(1 + 2^{-n}\big)^3 \big(1 - 2^{-n+1}\big) \geq \big(1 - 2^{-n}\big)^3 \big(1 + 2^{-n+1}\big)$$

and hence true for $n > 1$.
Under the induction assumption $e_{n-k} \leq e_n + 2 \sum_{j=n-k+1}^n e_j$ the induction assertion $e_{n-k-1} \leq e_n + 2 \sum_{j=n-k}^n e_j$ is equivalent to $e_{n-k-1} \geq 3\, e_{n-k}$ that has been verified in the induction start.

qed

8 Computation of Zeroes

The numerical determination of zeroes is indispensable if the zeroes of the first derivative of some function cannot determined analytically in order to compute for example extreme values of that function.

For example, this is the case for higher order polynomials.

8.1 Computation of Zeroes per Intervall Bisection

The field for the functions values is deliberately scaled 'too small' because in this method only the signs of function values are relevant:
A start intervall $[a, b]$ with the invariant $f(a)f(b) < 0$ is bisected by its middle point $m = (a+b)/2$. It is substituted by its right or left half intervall satisfying the invariant.

$f(x) =$ get f

i.e. $f(x) =$

$a =$	$f(a) =$	$[a, b]$ halbieren
$b =$	$f(b) =$	

n= m= $f(m) =$

test reset

Ex.: Termination critera? start intervall?

Ex.: Compute extrema of functions.

8.2 Computation of Zeroes per regula falsi

The start intervall $[a, b]$ with the invariant $f(a)f(b) < 0$ is devided by the zero $m = a - f(a)\frac{b-a}{f(b)-f(a)}$ of the secant line. It is substituted by the right or left intervall satisfying the invariant.

$f(x) =$ get f

i.e. $f(x) =$

$a =$ $f(a) =$ regula falsi

$b =$ $f(b) =$

n= $m =$ $f(m) =$

test reset

Ex.: Termination critera? comparison to interval subdivision?

Ex.: Compute extrema of functions.

8.3 Computation of Zeroes per Newton Method

It is assumed that the conditions for convergence of the Newton-Raphson method are fulfilled: the zero is approximated by the zeroes $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ of the tangent line in $(x_n, f(x_n))$.

$f(x) =$ get f

i.e. $f(x) =$

$f'(x) =$ get f'

i.e. $f'(x) =$

$x_{n+1} =$ $f(x_{n+1}) =$ Newton

$x_1 =$ $x_{n+1} - x_n =$ test

$n =$ 0 $f(x_n) =$ reset

Ex.: Compute $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$, π, $\pi/4$ and the like.

Ex.: Compute extrema of functions.

Ex.: Termination critera? Comparison to regula falsi?

9 Optimization

Numerical optimization determines minima of some function $f(x)$. Maxima of f are minima of $-f$.

9.1 Golden Section Search

As with interval bisection we have to know that in some interval $[a, b]$ the function f has exactly one minimum x^*. Then f is mononically decreasing left of x^* and monotonically increasing right of x^*. Specifically, if $f(x_1) < f(x_2)$ for some $a < x_1 < x_2 < b$, then $x^* \notin (x_2, b]$, hence $x^* \in [a, x_2)$, and if $f(x_1) > f(x_2)$ for some $a < x_1 < x_2 < b$, then $x^* \notin [a, x_1)$, hence $x^* \in (x_1, b]$. Thus the interval containing the minimum steadily is shortened. Highest efficiency is achieved if the function has to be evaluated only once in each step and if the location of x_1 and x_2 relative to a and b is constant and invariant, namely $x_1 = a + (1 - \tau)(b - a)$ and $x_2 = a + \tau(b - a)$ for $\tau = (\sqrt{5} - 1)/2$.

$f(x) =$ get f

i.e. $f(x) =$

a= b= test reset golden section

n=	$a =$		$f(a) =$	
	$x_1 =$		$f(x_1) =$	
	$x_2 =$		$f(x_2) =$	
	$b =$		$f(b) =$	
	min $\approx \frac{a+b}{2} =$		$f(\frac{a+b}{2}) =$	

Ex.: Compute minima of functions like $f(x) = e^{-x} \sin x$ or $g(x) = (x - 1)^n$ etc.

Ex.: Termination critera?

9.2 Optimization per Newton

Why not simply determine a zero of the first derivative using the Newton algorithm? Of course, then we have to have the first and second derivatives of the function to be minimized.

$f(x) =$		get f f' f''
$f'(x) =$		
$f''(x) =$		
i.e. $f(x) =$		
i.e. $f'(x) =$		
i.e. $f''(x) =$		

$x_o =$ test reset Newton

$n =$ $x =$ $f(x) =$

Ex.: Compute minima of functions like $f(x) = e^{-x} \sin x$ or $g(x) = (x-1)^n$ etc.

Ex.: Termination critera?

Ex.: Searching for zeros of the first derivative not necessarily results in finding minima! Find examples.

10 Integration

Numerical integration for $\int_a^b f(x)\,dx$ is a last resort if there is no antiderivative in closed form.

To determine the complexity of the competing methods the number of evaluations of the function to be integrated is used.

10.1 Integration per Trapezoidal Rule

The integral is approximated by the sum of the areas of certain trapezoids.

$f(x) =$ get f

i.e. $f(x) =$

a= b= $\int_a^b f(x)\,dx \approx I_n =$

Trapez

n= #(f(x))= $|I_n - I_{n-1}| =$

reset

Ex.: Compute known integrals like $\int_0^\pi \sin x\,dx$ or $\int_1^2 x^{-2}\,dx$ etc.

Ex.: Termination critera? Optimisation by doubling instead of incrementing n ?

10.2 Integration per Simpson Rule

The function to be integrated is approximated piecewise by parabolas. The sum of the integrals of these parabolas then approximates the integral.

n (even) is incremented by 2.

$f(x) =$ get f

i.e. $f(x) =$

a= b= $\int_a^b f(x)\,dx \approx I_n =$

Simpson

n= #(f(x))= $|I_n - I_{n-1}| =$

reset

Ex.: Compute integrals of second degree polynomials. Why is the approximation correct for each n ?

Ex.: Compute integrals like $\int_1^e \frac{1}{x}\,dx$ etc.

Ex.: Comparison with the trapezoidal rule? Generalisations?

11 1^{st} Order Ordinary Differential Equations

The methods of Euler, Heun, Euler-Cauchy and Runge-Kutta show how first order ordinary differential equations are solved numerically. Of course, this is relevant if there is no known analytical solution in closed form.

Given the first order inital value problem

$$\boxed{y' = f(x,y) \text{ with } y(x_o) = y_o}.$$

To solve such a problem, the function $f(x,y)$, the initial condition, x_o and y_o, as well as the number n of increments have to be specified in order to approximate the function value $y_n \approx y(x_n)$ in x_n. Then, the results of the different methods can be compared. If the exact solution $y = y(x)$ is available the quality of these results can be assessed.
The test button saves to input the differential equation $y' = y$ with initial condition $y(0) = 1$ and solution $y(x) = e^x$.

Some classical methods to solve differential equations numerically are presented, namely the methods of Euler, Heun, Euler-Cauchy and the classical version of the Runge-Kutta method.

11.1 Euler-Method

The Euler-method approximates the function by the tangent line in the argument computed at last:

$$\boxed{y(x_{i+1}) \approx y_{i+1} = y_i + h\, f(x_i, y_i) \text{ with } y(x_o) = y_o}$$

11.2 Heun-Method

The Heun Method is a predictor-corrector method: y_{n+1} is computed by the secant line through (x_n, y_n) whose slope is the average (corrected) of the slope in x_n and an slope $f(x_{n+1}, \bar{y}_{n+1})$ in $(x_{n+1}, \bar{y}_{n+1})$ (predicted):

$$\boxed{y(x_{i+1}) \approx y_{i+1} = y_i + \frac{h}{2}\big(f(x_i, y_i) + f(x_{i+1}, y_i + h\, f(x_i, y_i))\big) \text{ with } y(x_o) = y_o}$$

11.3 Euler-Cauchy-Method

The Euler-Cauchy-Method uses an approximation of the slope of the tangent line in the midpoint $x_{n+1/2} = (\frac{1}{2}x_n + x_{n+1})$ of $[x_n, x_{n+1}]$,
hence $y(x_{n+1/2}) \approx y_{n+1/2} = y_n + \frac{h}{2} f(x_n, y_n)$.

$$\boxed{y(x_{i+1}) \approx y_{i+1} = y_i + h\, f\big(x_{i+1/2}, y_i + \frac{h}{2} f(x_i, y_i))\big) \text{ with } y(x_o) = y_o}$$

11.4 Runge-Kutta-Method

The classical Runge-Kutta method approximates the unknown function $y(x)$ in x_i by a secant line whose slope is the weighted sum of the four slopes y'_i at x_i, $\bar{y}'_{i+1/2}$ and $\bar{\bar{y}}'_{i+1/2}$ at $x_{i+1/2}$ as well as $\bar{y}'_{i+1}$ at x_{i+1}.

$$\boxed{\begin{array}{llllll} y_{i+1} & = & y_i + \frac{h}{6}\big(y'_i + 2\bar{y}'_{i+1/2} + 2\bar{\bar{y}}'_{i+1/2} + \bar{y}'_{i+1}\big) & & & \text{where} \\ y'_i & = & f(x_i, y_i) & \bar{y}'_{i+1/2} & = & f(x_i + \frac{h}{2}, y_i + \frac{h}{2} y'_i) \\ \bar{\bar{y}}'_{i+1/2} & = & f(x_i + \frac{h}{2}, y_i + \frac{h}{2} \bar{y}'_{i+1/2}) & \bar{y}'_{i+1} & = & f(x_i + h, y_i + h\bar{\bar{y}}'_{i+1/2}) \end{array}}$$

11.5 Interactive Synopsis of these Methods

$f(x, y) =$ get f

i.e. $f(x, y) =$

$y(x) =$ get y

i.e. $y(x) =$

$x_o =$	0	Euler $y =$	
$y_o =$	1	Heun $y =$	
$x_n =$	1	Euler-Cauchy $y =$	
$n =$	1	Runge-Kutta $y =$	

exact $y(x) = y($ $) =$

test step repeat reset

Ex.: Experiment with $y' = \cos x$ and $y(0) = 0$ or $y' = xy$ and $y(0) = 1$ or the like.

Ex.: Compare the different methods by their complexity in terms of number of evaluations of $f(x, y)$.

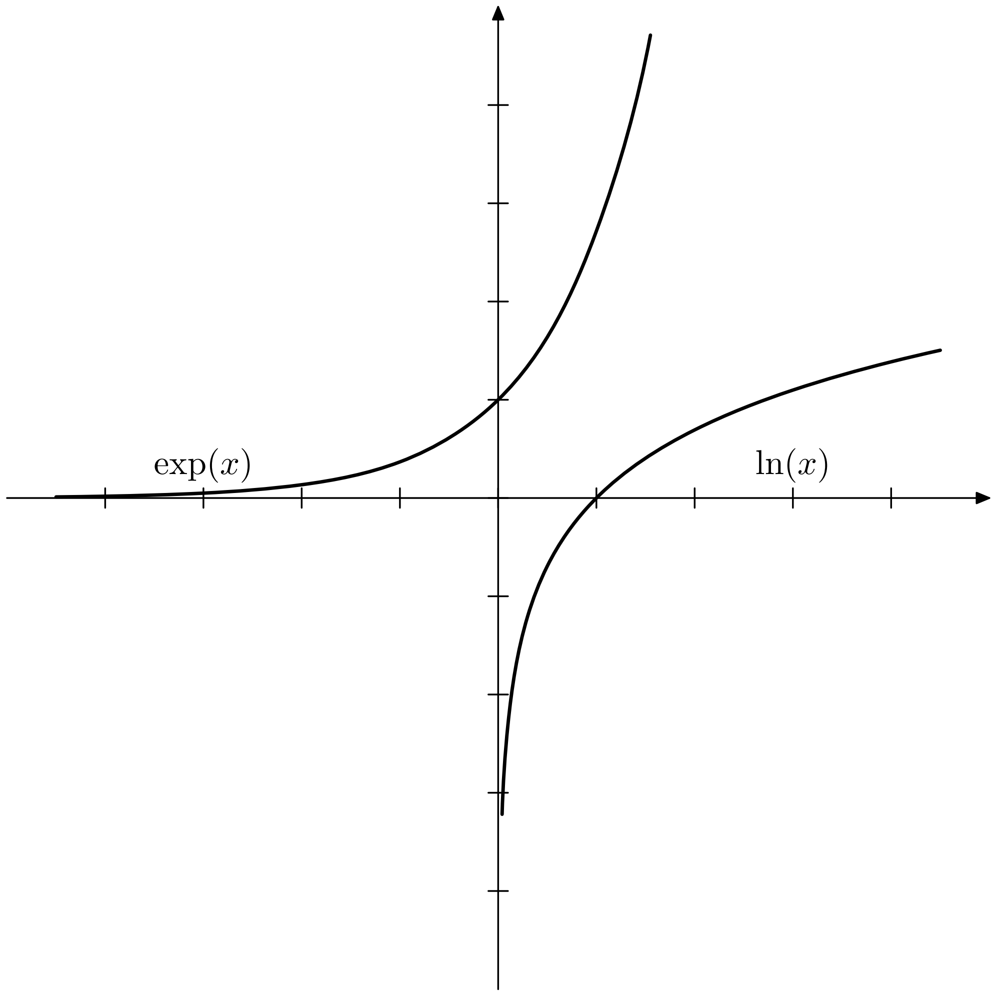
11.6 Systems of Ordinary First Order Differential Equations

The methods of the previous sections can be applied to systems of ordinary first order differential equations also. The default example is the development of predator and prey populations in time.

nyi

A Graphs of Elementary Functions

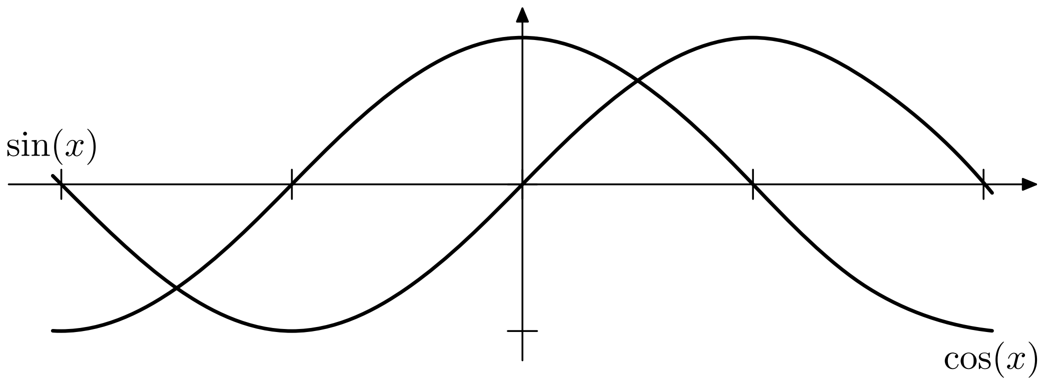
A.1 Exponential Function and Logarithm



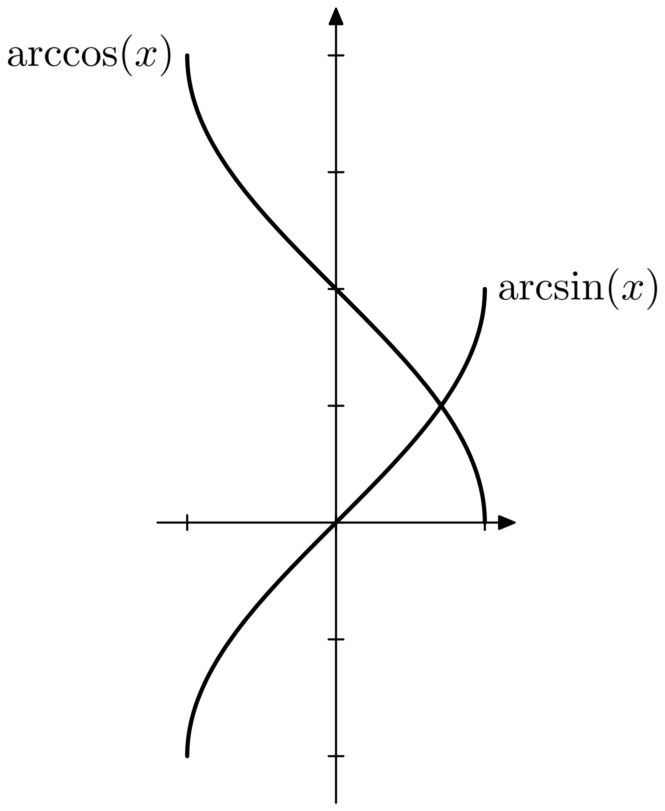
exp ln

Ex.: Label the ticks and check functions properties.

A.2 Sine and Cosine and their Inverses

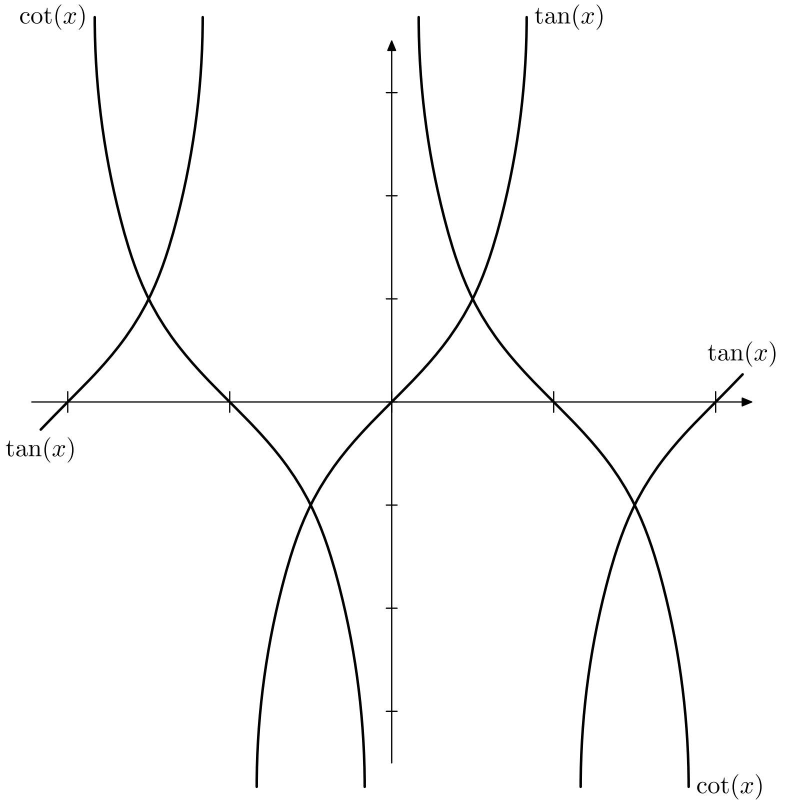


sin cos

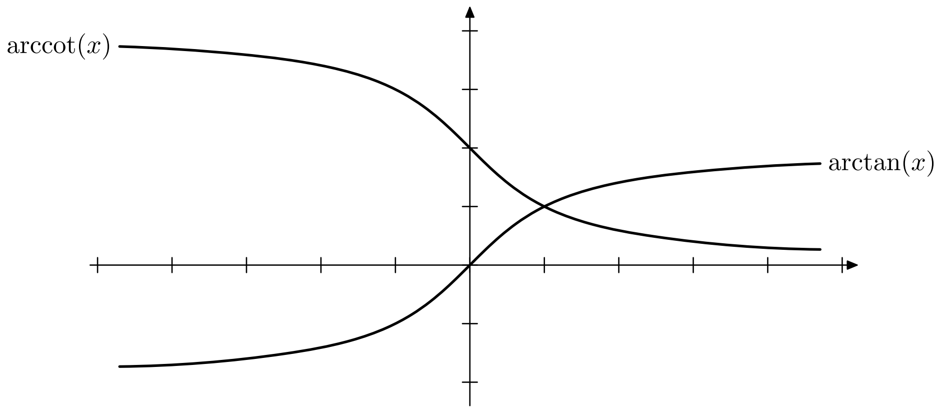


Ex.: Label the ticks and check functions properties.

A.3 Tangent and Cotangent and their Inverses

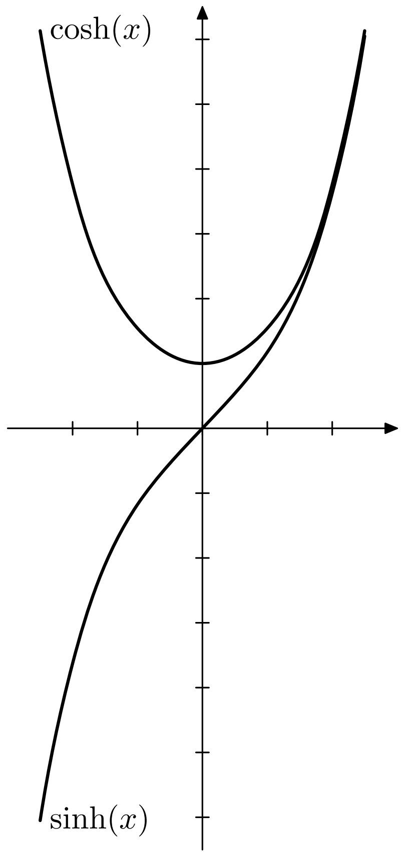


tan cot

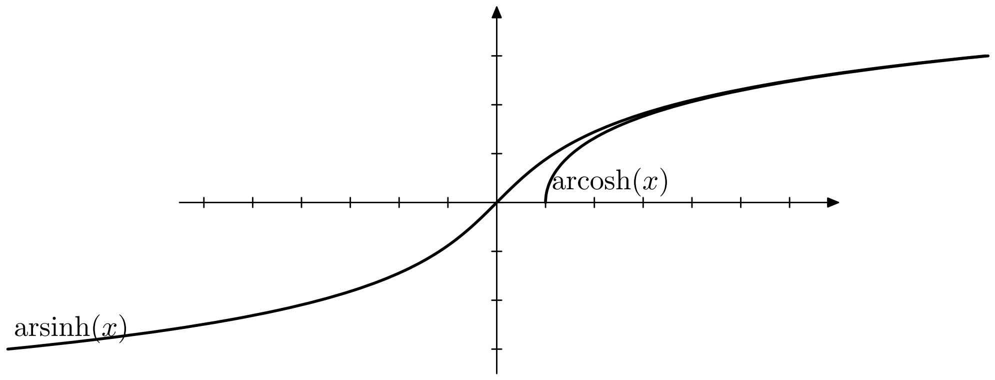


Ex.: Label the ticks and check functions properties.

A.4 Hyperbolic Sine and Cosine and their Inverses

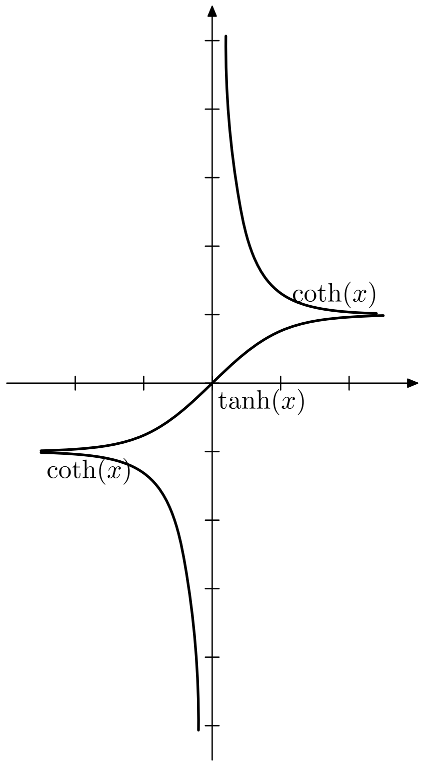


sinh cosh

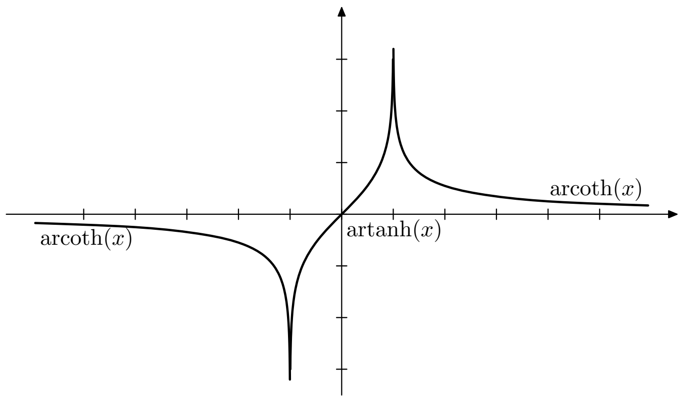


Ex.: Label the ticks and check functions properties.

A.5 Hyperbolic Tangent and Cotangent and their Inverses



tanh coth



Ex.: Label the ticks and check functions properties.

Contents