Numerik – interaktiv

Thomas Risse
Institut für Informatik und Automation
Hochschule Bremen

5. Dezember 2007

Zusammenfassung

Dieses Dokument erlaubt, diverse grundlegende numerische Algorithmen interaktiv zu untersuchen.
So bestimmt dieses Dokument die Berechnungsgenauigkeit, ermöglicht einfachste 2D und 3D Vektorrechnung, löst Systeme linearer Gleichungen, invertiert Matrizen, listet einige numerische Konstanten, bietet die Möglichkeit, beliebige aus elementaren Funktionen zusammengesetzte Funktionen auszuwerten, stellt diese elementaren Funktionen zusammen mit ihren Umkehr-Funktionen vor, führt CORDIC-Algorithmen ein, erlaubt, Nullstellen, Minima und Integrale solcher beliebigen Funktionen zu approximieren sowie gewöhnliche Differentialgleichungen erster Ordnung numerisch zu lösen.

1 Vorbemerkung

Dieses Dokument soll helfen, Einsichten in Arbeitsweise und Leistung einiger grundlegender numerischer Algorithmen zu vermitteln, indem es die Gelegenheit bietet, jeder Algorithmus direkt auszuführen, Abhängigkeit von Startwerten zu untersuchen, Komplexität von verschiedenen Algorithmen mit derselben Zielsetzung zu vergleichen usw.

1.1 Konventionen und Gebrauch

Im Folgenden grundsätzlich:

click 

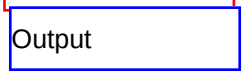
, um eine Operation auszuführen oder eine Funktion auszuwerten;

click 

, um Argumente und Ergebnis zu löschen,

Eingaben in 

-Felder

Ausgaben in 

oder 

-Felder

Das layout zielt darauf ab, alle für einen Algorithmus relevanten Informationen – Bildschirm-füllend, also Fenster-Breite (Strg-2) im Acrobat Reader oder Seiten-Breite (Strg-3) im Acrobat Writer – auf einer Seite darzustellen.

1.2 Rechengenauigkeit

Die *relative Rechengenauigkeit*[1] ν wird durch

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

bestimmt. Für JavaScript gilt: JS precision $\nu =$

Üb. Genauigkeit von JavaScript Berechnungen ausgedrückt in Anzahl von Dezimal-Ziffern?

1.3 Gleitpunkt-Arithmetik

Aufgrund der beschränkten Genauigkeit gelten bestimmte Gesetze der Arithmetik im Rechner **nicht**! etwa das der Assoziativität $(a+b)+c = a+(b+c)$

$a =$ 99999999999
$b =$ -99999999999
$c =$ 0.000005

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

reset Add

Üb. Berechne den relativen Fehler!

[1] Im Gegensatz zu JavaScript oder Java-applets können Computer-Algebra-Pakete wie etwa Mathematica, Maple oder MuPad mit beliebig vorzugebender Genauigkeit rechnen.

1.4 Gleitpunkt-Arithmetik mit gegebener Präzision

JavaScript-Berechnungen sind ziemlich genau. Um die Effekte der beschränkten Genauigkeit demonstrieren zu können, kann nun die Anzahl der Dezimal-Ziffern für die Darstellung von Gleitpunkt-Zahlen vorgegeben werden.

Argumente und Ergebnisse dargestellt mit JavaScript-Genauigkeit

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

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

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

$p = \#$ Dezimal-Ziffern = 3 test random reset Add

Argumente und Ergebnisse dargestellt mit p Dezimal-Ziffern

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

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

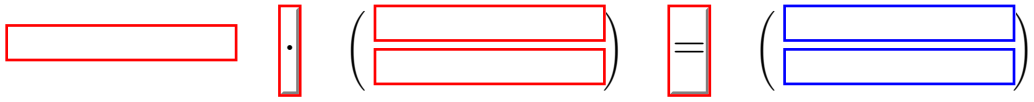
$\overline{(\bar{b} + \bar{c})} =$ $\overline{\bar{a} + \overline{(\bar{b} + \bar{c})}} =$

Üb. Darstellung der eingegebenen Zahlen? relativer Fehler für verschiedene p ?

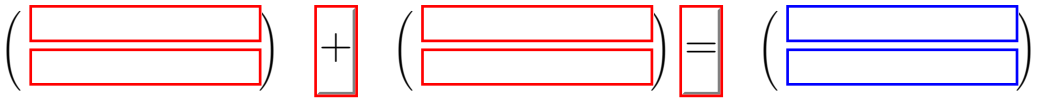
2 Vektor-Rechnung

2.1 Operationen auf Vektoren der Ebene

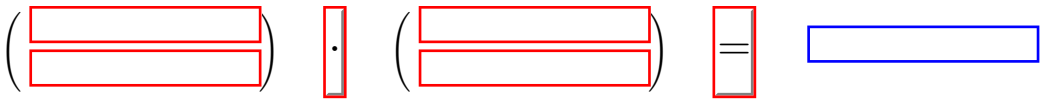
2.1.1 skalare Vielfache $c\,\vec{a}$ von Vektoren $\vec{a}$ der Ebene



2.1.2 Addition $\vec{a}+\vec{b}$ von Vektoren der Ebene

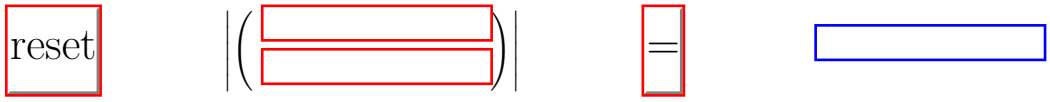


2.1.3 Skalar-Produkt $\vec{a}\cdot\vec{b}$ von Vektoren der Ebene



Üb.: Welchen Winkel $\angle(\vec{a},\vec{b})$ schließen die Vektoren $\vec{a} = \frac{\sqrt{2}}{2}(1,1)$ und $\vec{b} = (\sqrt{3}-2,1)$ ein? Verwende Abschnitt 5.

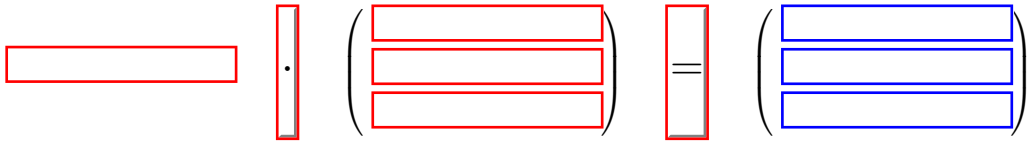
2.1.4 Länge oder Betrag $|\vec{a}|$ von Vektoren der Ebene



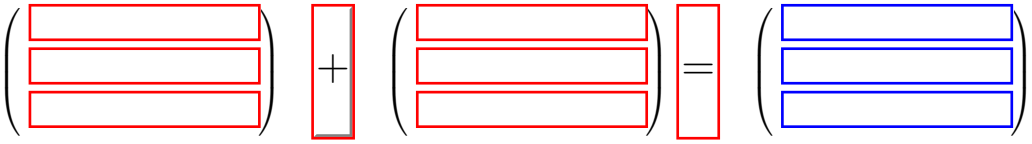
Üb.: Normalisiere Vektoren wie $\vec{a} = \frac{\sqrt{2}}{2}(1,1)$ oder $\vec{b} = (\sqrt{3}-2,1)$. Verwende Abschnitt 5.

2.2 Operationen auf Vektoren im Raum

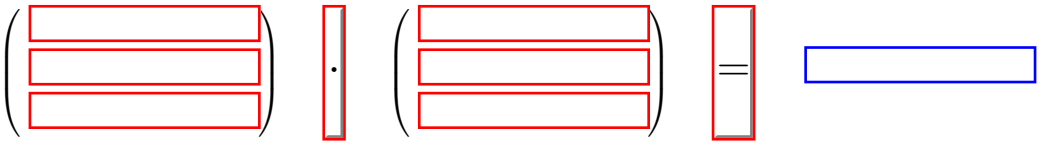
2.2.1 skalare Vielfache $c\vec{a}$ von Vektoren $\vec{a}$ im Raum



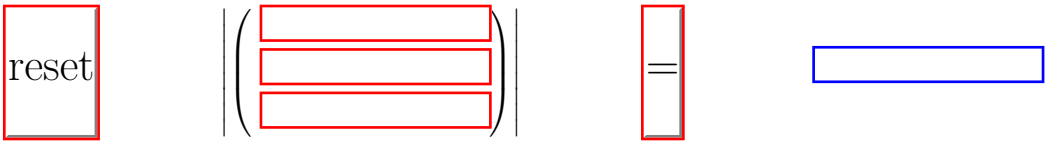
2.2.2 Addition $\vec{a}+\vec{b}$ von Vektoren im Raum



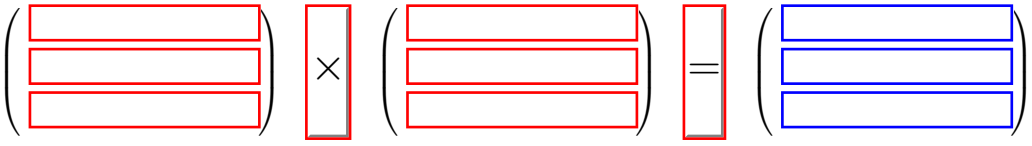
2.2.3 Skalar-Produkt $\vec{a}\cdot\vec{b}$ von Vektoren im Raum



2.2.4 Länge oder Betrag $|\vec{a}|$ von Vektoren im Raum



2.2.5 Vektor-Produkt $\vec{a}\times\vec{b}$ von Vektoren im Raum



Üb.: Verifiziere: $\vec{e}_x$, $\vec{e}_y$ und $\vec{e}_z$ sind orthonormiert.

Üb.: Konstruiere 'Einheitswürfel' mit Ecken in $\vec{0}$ und $\frac{\sqrt{3}}{3}(1,1,1)$.

3 lineare Gleichungssysteme

Lineare Gleichungssysteme sind durch $\mathbf{Ax} = \mathbf{b}$ gegeben. Zur Demonstration werden nur 3×3-Koeffizienten-Matrizen $\mathbf{A}$ verwendet, also LGSe mit drei Gleichungen in drei Unbekannten. Verglichen werden das Gauß'sche Eliminationsverfahren ohne und mit Pivotisierung und das Gauß-Seidel-Verfahren.

3.1 Gauß'sches Eliminationsverfahren

Koeffizienten-Matrix $\mathbf{A}$ und Vektor $\mathbf{b}$ der rechten Seite spezifizieren:

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

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

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

x_1 eliminieren dann x_2 eliminieren

x_3 bestimmen x_2 bestimmen x_1 bestimmen

Mit originaler Koeffizienten-Matrix $\mathbf{A}$, Lösungsvektor $\mathbf{x}$ und originalem Vektor $\mathbf{b}$ der rechten Seite berechne Residuum $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ Residuum

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

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

Üb.: Genauigkeit? Lösungsbedingungen? unterbestimmte Systeme linearer Gleichungen?

3.2 Gauß'sches Eliminationsverfahren mit Pivotisierung

Details zur partiellen und vollständigen Pivotisierung siehe z.B.
`www.weblearn.hs-bremen.de/risse/MAI/docs/heath.pdf`

Koeffizienten-Matrix $\mathbf{A}$ und Vektor $\mathbf{b}$ der rechten Seite spezifizieren:

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

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

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

Mit originaler Koeffizienten-Matrix $\mathbf{A}$, Lösungsvektor $\mathbf{x}$ und originalem Vektor $\mathbf{b}$ der rechten Seite berechne Residuum $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ Residuum

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

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

Üb.: Was passiert, wenn die Matrix schlecht konditioniert ist? vgl.
`www.weblearn.hs-bremen.de/risse/MAI/docs/heath.pdf`

3.3 Stifel'sches Verfahren – LGS & Matrix Inversion

Das *Stifel'sche Austauschverfahren* löst quadratische Systeme linearer Gleichungen $\mathbf{Ax} = \mathbf{b}$, indem jede Gleichung nach einer Unbekannten aufgelöst und die Unbekannte in allen anderen Gleichungen durch den sich ergebenden Ausdruck ersetzt wird. Dabei werden also *Unbekannte gegen Konstanten (der rechten Seite) ausgetauscht*. Das Verfahren löst $\mathbf{Ax} = \mathbf{b}$, indem die Koeffizienten-Matrix $\mathbf{A}$ invertiert wird, so daß sich $\mathbf{x} = A^{-1}\mathbf{b}$ ergibt.

Austausche werden im folgenden Schema durchgeführt. Die sogenannte *Keller-Zeile* speichert Zwischenergebnisse zur Wiederverwendung.

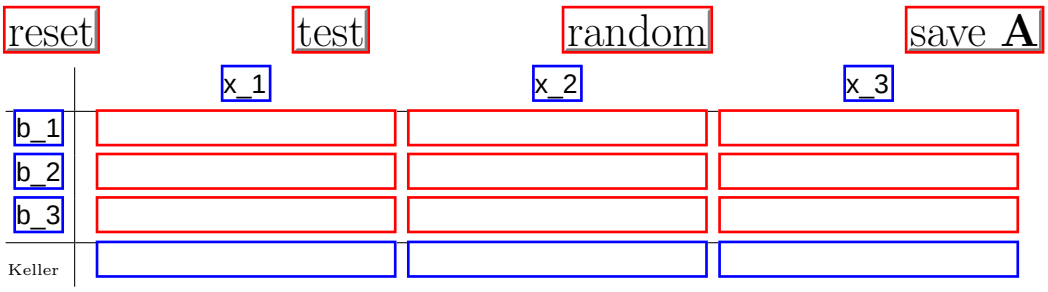
	x_1	x_2	$\dots$	x_j	$\dots$	x_n
b_1	a_{11}	a_{12}	$\dots$	$\underline{a_{1j}}$	$\dots$	a_{1n}
$\vdots$				$\vdots$		
b_i	$\underline{a_{i1}}$	$\underline{a_{i2}}$	$\dots$	$\underline{\underline{a_{ij}}}$	$\dots$	$\underline{a_{in}}$
$\vdots$				$\vdots$		
b_{n-1}	$a_{n-1,1}$	$a_{n-1,2}$	$\dots$	$\underline{a_{n-1,j}}$	$\dots$	$a_{n-1,n}$
b_n	a_{n1}	a_{n2}	$\dots$	$\underline{a_{nj}}$	$\dots$	a_{nn}
Keller				—		

b_i wird gegen x_j durch die folgende Prozedur ausgetauscht:

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

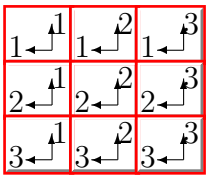
austauschen a_{ij} `= 1/pivot; for (`$k \neq i$`)` a_{kj} `/= pivot;`
`for (`$k \neq j$`)` a_{kj}`=tmp[k];`
`for (`$u \neq i$`) for (`$v \neq j$`)` a_{uv} `+=` a_{uj}`*tmp[v];`

Üb.: Verifiziere die Invarianz von *Spalten-Vektor (linke Seite des Schemas) = Matrix mal transponierter Zeilen-Vektor (oben im Schema).*



mit $\det(\mathbf{A}) =$

1. click: Keller füllen
2. click: Rest setzen
3. click: Pivot-Spalte setzen
4. click: Pivot-Zeile setzen



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

$$\mathbf{A}\,\mathbf{A}^{-1} = \left(\right)$$

$$\left(\right)$$

$$\approx \left(\right)$$

Üb.: Genauigkeit? Auswirkungen verschiedener Reihenfolgen der Austauschschritte?

3.4 Gauß-Seidel-Verfahren

Vorausgesetzt, alle Hauptdiagonal-Elemente der Koeffizienten-Matrix sind von Null verschieden, d.h. $a_{ii} \neq 0$ für $i = 1, \ldots, n$, so läßt sich $\mathbf{Ax} = \mathbf{b}$ nach den Unbekannten der Hauptdiagonalen auflösen, für $n = 3$ also

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

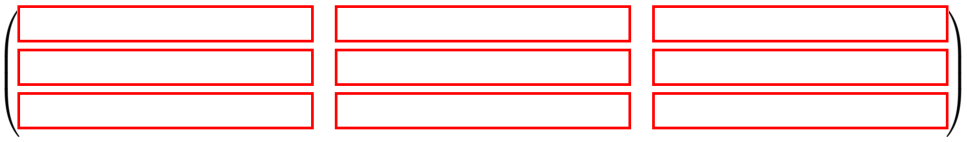
und per

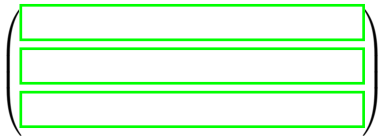
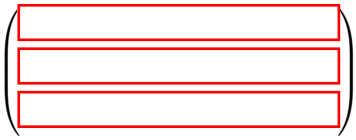
$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{a_{11}} \left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}\right) \\ x_2^{(k+1)} &= \tfrac{1}{a_{22}} \left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)}\right) \\ x_3^{(k+1)} &= \tfrac{1}{a_{33}} \left(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)}\right) \end{aligned}$$

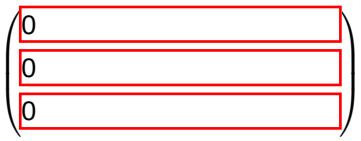
dazu nutzen, ausgehend von einem Start-Vektor $\mathbf{x}^{(0)}$ schrittweise $\mathbf{x}^{(k)}$ einzusetzen und unter Verwendung bereits verbesserter Komponenten weiter zu verbessern und so $\mathbf{x}^{(k)}$ in $\mathbf{x}^{(k+1)}$ zu überführen.

Das Verfahren konvergiert, wenn etwa (gegebenenfalls nach Umordnung) die Hauptdiagonalelemente jeweils die übrigen Elemente der zugehörigen Zeile dominieren, d.h. wenn $|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}|$ für $i = 1, \ldots, n$ gilt.

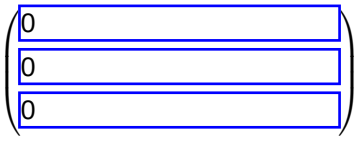
Koeffizienten-Matrix $\mathbf{A}$ und Vektor $\mathbf{b}$ der rechten Seite spezifizieren:

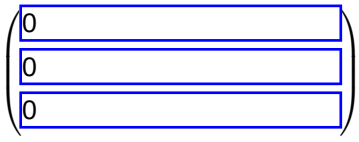
$\mathbf{A} =$ 

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

Start-Vektor $\mathbf{x}^{(0)} =$  , $\det(\mathbf{A}) =$

reset | test | random $\mathbf{A}$ | random $\mathbf{b}$ | random $\mathbf{x}^{(0)}$ | Konvergenz?

$k =$ $\mathbf{x}^{(k)} =$ 

$k+1 =$ $\mathbf{x}^{(k+1)} =$ 

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

Üb.: Abhängigkeit vom Start-Vektor? Konvergenzgeschwindigkeit?

Üb.: Vergleich direktes (Gauß) mit iterativem (Gauß-Seidel) Verfahren?

4 einige Konstanten

$e =$ $\pi =$

$\ln 2 =$ $\ln\ 10 =$

$\mathrm{ld}\ e = \log_2 e =$ $\lg\ e = \log_{10} e =$

$\sqrt{1/2} =$ $\sqrt{2} =$

get constants

Üb.: Vergleiche die 'eingebauten' Konstanten mit berechneten Werten geeigneter Funktionen im Abschnitt 5.

5 Berechnung von Funktionswerten

Ausgewertet werden beliebige, aus `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` und den Konstanten `e` und `pi` (genauso geschrieben) zusammengesetzte Funktionen in der einen unabhängigen Variablen `x`. Dabei ist die charakteristische Funktion `chi(x,a,b)` durch $\chi_{[a,b]}(x) = \begin{cases} 1 & x \in [a,b] \\ 0 & \text{sonst} \end{cases}$ mit $a < b$ definiert. Auch für das Argument `x` selbst sind beliebige **konstante** Ausdrücke in den obigen Funktionen und Konstanten zugelassen.

$f(x) =$

i.e. $f(x) =$

f( )= test

Üb.: Berechne $\sin(x)$ und $\cos(x)$ etwa für $x = 30^o, 45^o, 60^o, \ldots$ mithilfe geeigneter Funktionen sin und cos.

Üb.: Bestimme $\arctan(x)$ für $x = 1, \sqrt{3}, \ldots$ in Grad.

Üb.: Werte Funktionen wie $\text{arsinh}(x)$ oder $\text{arcosh}(x)$ für $x = 0, 1, \ldots$ aus.

Üb.: Was passiert mit $\frac{\sin(x)}{x}$ für $0 < x \ll 1$, wenn $\lim_{x \to 0} \frac{\sin(x)}{x}$ zu bestimmen ist.

Üb.: Vergleiche `gamma(n)` und `factorial(n-1)`.

6 elementare Funktionen mit Inversen

6.1 Exponentialfunktion und Logarithmus

6.1.1 Exponentialfunktion

$\exp(\quad) =$

$\quad = \ln(\quad)$

Graph

6.1.2 Logarithmus

$\ln(\quad) =$

$\quad = \exp(\quad)$

Graph

Üb.: Berechne $e^{\ln c}$ für $c > 0$ und $\ln e^{d}$ für beliebige reelle d.

6.2 trigonometrische Funktionen mit ihren Umkehr-Funktionen

6.2.1 Sinus und Arcus-Sinus

$\sin($ arcus $\square$ degree $\square$ $) = \square$

$\square = \arcsin(\square)$

Graph

6.2.2 Cosinus und Arcus-Cosinus

$\cos($ arcus $\square$ degree $\square$ $) = \square$

$\square = \arccos(\square)$

Graph

6.2.3 Tangens und Arcus-Tangens

$\tan($ arcus $\square$ degree $\square$ $) = \square$

$\square = \arctan(\square)$

Graph

6.2.4 Cotangens und Arcus-Cotangens

$\cot($ arcus $\square$ degree $\square$ $) = \square$

$\square = \operatorname{arccot}(\square)$

Graph

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

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

6.3 hyperbolische Funktionen mit ihren Umkehr-Funktionen

6.3.1 Sinus hyperbolicus

sinh( ) =

=arsinh( )

Graph

6.3.2 Cosinus hyperbolicus

cosh( ) =

=arcosh( )

Graph

6.3.3 Tangens hyperbolicus

tanh( ) =

=artanh( )

Graph

6.3.4 Cotangens hyperbolicus

coth( ) =

=arcoth( )

Graph

Üb.: Welche Asymptoten haben tanh und coth? Welche Problematik ergibt sich für artanh(tanh(x)) bzw. für arcoth(coth(x)) ?

Üb.: Die Implementierung von sinh, cosh, tanh und coth ist 'straight forward'. *Recherchiere:* Wie sind aber deren Inverse arsinh, arcosh, artanh und arcoth zu implementieren, wenn allein `ln` und `sqrt` zur Verfügung stehen?

7 CORDIC – Auswertung elementarer Funktionen in hardware

CORDIC steht für *¨COordinate Rotation DIgital Computer¨*. CORDIC-Algorithmen berechnen bestimmte elementare Funktionen unter fast ausschließlicher Verwendung von schnellen Operationen, nämlich Additionen (*adds*) und Multiplikationen mit Zweier-Potenzen (*shifts*).
CORDIC-Algorithmen sind daher für Implementierungen in (Fest-Punkt) hardware ausgezeichnet geeignet.

Die Rotation von Vektoren der (komplexen) Ebene um den Winkel φ mit dem Ursprung als Fixpunkt ist gegeben durch

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

Insbesondere für $\varphi = \pm\arctan(2^{-i})$ und damit für $\tan\varphi = \pm 2^{-i}$ gilt

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

Bis auf Multiplikation mit $\frac{1}{\sqrt{1+2^{-2i}}}$ ist die Rotation um $\arctan 2^{-i}$ also allein mit *adds* und *shifts* zu berechnen. Hintereinanderausführung liefert

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

wobei $\delta_i = \pm 1$ die Richtung der i-ten Rotation vorgibt und für den *gain* g_∞ des Algorithmus gilt

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

Üb.: Formuliere den obigen Algorithmus unter alleiniger Verwendung der Additionstheoreme von Sinus und Cosinus.

7.1 trigonometrische Funktionen (circular $m = 1$, rotating)

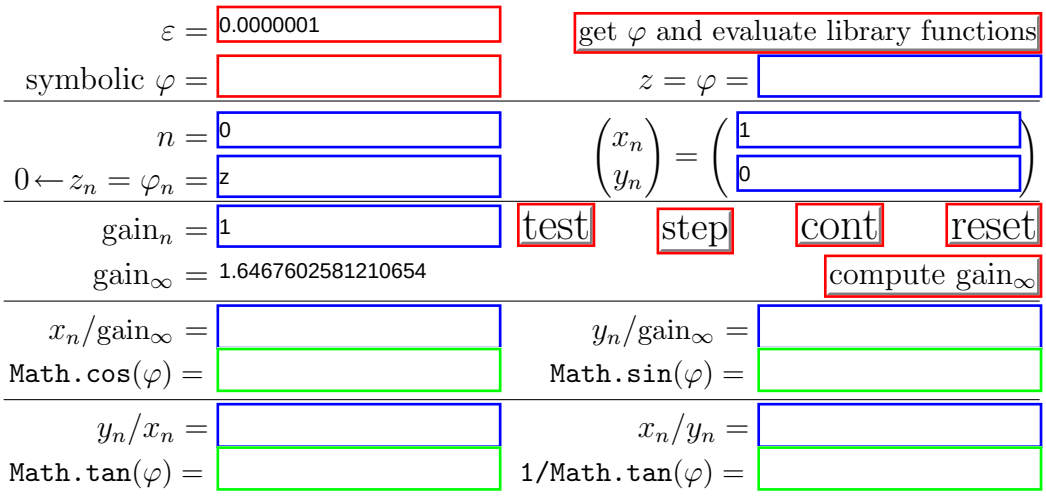
Die Vektoren $\vec{z}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ mit $z_o = \vec{e}_x$ approximieren den Vektor $g \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix}$, wenn die Rotation um φ durch eine Folge von Rotationen um $\pm \arctan 2^{-i}$ approximiert wird. Der folgende Algorithmus approximiert $\sin\varphi$ und $\cos\varphi$ für φ im Bogenmaß mit $|\varphi| < \frac{\pi}{2}$. Dabei ist `arctan(k)` etwa durch eine *look up table* zu realisieren.

```
// 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)
```

Üb.: Entwirf die look up table für `atan(kk)` !

Üb.: Welcher systematische Fehler ist hinzunehmen, sollen nicht weitere Multiplikationen (außer den beiden Divisionen durch g am Schluß) notwendig werden?

`Math.sin`, `Math.cos` und `Math.tan`, die von JavaScript zur Verfügung gestellten Bibliotheksfunktionen, dienen als Referenzen (in grün). 'Symbolische' $\varphi \in [-\frac{\pi}{2}, \frac{\pi}{2}] \approx [-1.57, 1.57]$ wie z.B. `pi/5`, `0.5` oder `arcsin(0.5)` eingeben!



Üb.: Welche Genauigkeit erzielt jeder Schritt des CORDIC-Algorithmus'?

Üb.: Was passiert, wenn diese Version des CORDIC-Algorithmus' auf Argumente wie $\frac{\pi}{4}$ angewandt wird? Welches sind die anderen kritischen Argumente?

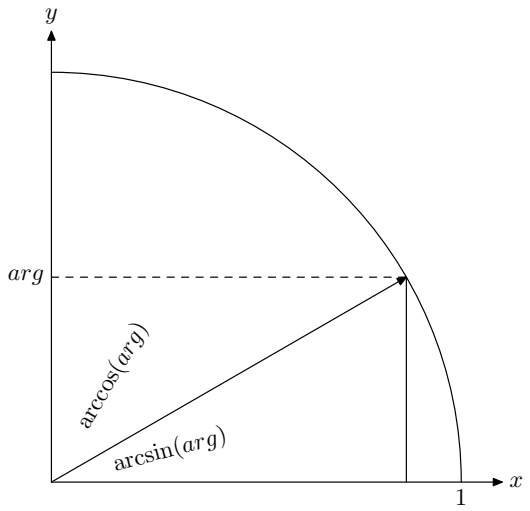
7.1.1 Polar- in Cartesische Koordinaten transformieren

Mit der gleichzeitigen Berechnung von Sinus und Cosinus können zu einem Vektor in Polar-Koordinaten (r, φ) auch die zugehörigen Cartesischen Koordinaten $(x, y) = r(\cos\varphi, \sin\varphi)$ bestimmt werden, indem der Cartesische Vektor $(r, 0)$ um den Winkel φ in den Ziel-Vektor $r(\cos\varphi, \sin\varphi)$ gedreht wird.

7.2 inverse trigonometrische Funktionen (circular $m = 1$, vectoring)

Um etwa $\varphi = \arcsin(arg)$ zu berechnen, wird der Einheitsvektor $\vec{e}_x$ einer Folge von Rotationen unterworfen, bis die y-Koordinate des gedrehten Vektors mit arg übereinstimmt. Dann kann $\arccos(arg)$ bestimmt werden aus

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



`Math.asin`, `Math.acos` und `Math.atan` sind von JavaScript zur Verfügung gestellte Bibliotheksfunktionen. Zulässige Argumente a von Arcus Sinus und Arcus Cosinus liegen in $[-1, 1]$, z.B. $a =$`sin`(0.5) oder $a =$`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 Cartesische in Polar-Koordinaten transformieren

Aus Cartesischen Koordinaten (x, y) sind die Polar-Koordinaten (r, φ) mit $r = \sqrt{x^2 + y^2}$ und $\varphi = \arctan \frac{y}{x}$ zu bestimmen: beides wird vom CORDIC-Algorithmus zur Berechnung des Arcus Tangens geliefert, der den Start-Vektor (x, y) in den Cartesischen Vektor $(r, 0)$ dreht. Da dabei die Länge erhalten bleibt, gilt $r = \sqrt{x^2 + y^2}$. Der Drehwinkel selbst ist ja $\varphi = \arctan \frac{y}{x}$.

7.3 hyperbolische Funktionen (hyperbolic $m = -1$, rotating)

Die Rotation der (komplexen) Ebene basiert auf den Additionstheoremen von Sinus und Cosinus. Für die hyperbolischen Funktionen sinh und cosh gelten die folgenden (entsprechenden) Additionstheoreme

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

Diese geben Anlaß, $\cosh a$ und $\sinh a$ für gegebenes Argument a durch eine Folge von 'hyperbolischen' Rotationen zu berechnen

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

Für geeignete $y = \operatorname{artanh} 2^{-i}$ bzw. $\tanh y = 2^{-i}$ können diese Rotationen (bis auf die Skalierung am Ende) ausschließlich mit shifts und adds ausgeführt werden.

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

wobei wegen $\frac{1}{\cosh y} = \sqrt{\frac{\cosh^2 y - \sinh^2 y}{\cosh^2 y}} = \sqrt{1-\tanh^2 y}$ für $y = \pm\operatorname{artanh} 2^{-i}$ gerade $\cosh(\pm\operatorname{artanh} 2^{-i}) = \frac{1}{\sqrt{1-2^{-2i}}}$ gilt.

Um zugleich $\cosh a$ und $\sinh a$ zu berechnen, muß a nur als Summe $a = \sum d_i \operatorname{artanh} 2^{-i}$ geeigneter 'Drehwinkel' $\pm\operatorname{artanh} 2^{-i}$ und 'Drehrichtungen' $d_i \in \{1, -1\}$ dargestellt werden. Ausgehend von $\cosh 0 = 1$ und $\sinh 0 = 0$ ergeben sich dann $\cosh a$ und $\sinh a$ als Folge solcher 'Rotationen'

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

Für den *gain* g_∞ des Algorithmus gilt

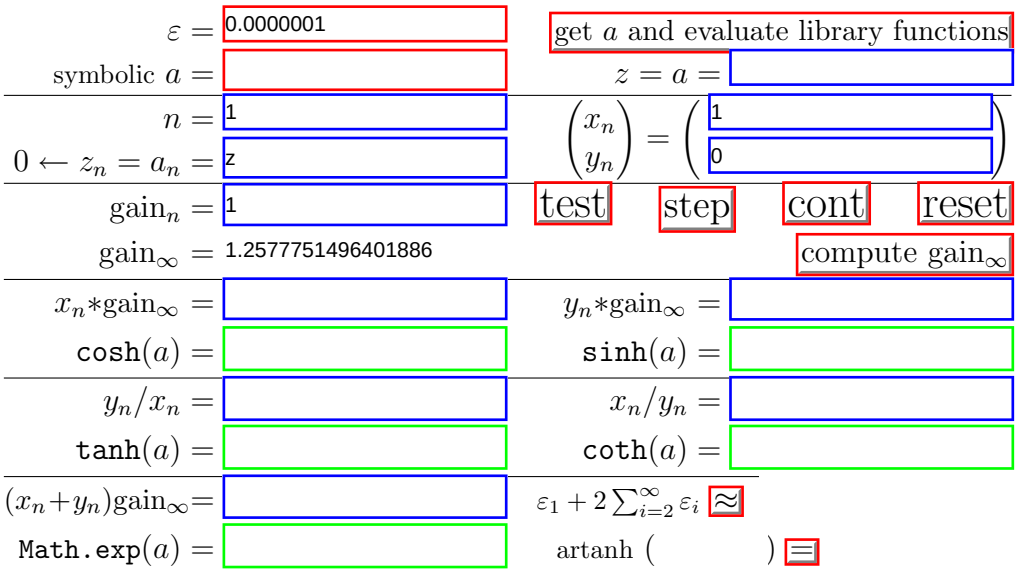
$$g_\infty = \prod_{i=0}^{\infty} \frac{1}{\sqrt{1-2^{-2i}}} = \lim_{n\to\infty} g_n = \lim_{n\to\infty} \prod_{i=0}^{n} \frac{1}{\sqrt{1-2^{-2i}}} \approx$$

```
g=0.6; k=1; // k= 2^-i
x=1; y=0;   // 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))
```

Üb.: Entwirf die look up table für `artanh(kk)` !

Üb.: Welcher systematische Fehler ist hinzunehmen, sollen nicht weitere Multiplikationen (außer den beiden Divisionen durch g am Schluß) notwendig werden?

Die lokal verfügbaren Funktionen `cosh`, `sinh`, `tanh` und `coth` sowie die JavaScript-Bibliotheksfunktion `Math.exp` dienen als Referenz (in grün).



Üb.: Welche Argumente im obigen Algorithmus sind zulässig?

Üb.: Wie ist mit unzulässigen Argumenten zu verfahren? Stellen Sie dazu ein beliebiges Argument $x \in \mathbb{R}$ als Summe eines zulässigen Argumentes z mit $|z| \leq 1$ und einem Vielfachen von $\ln 2$ dar und verwenden Sie die Additionstheoreme. Mit $x = z + m \ln 2$ gilt dann

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

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

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

7.3.1 Exponential-Funktion und andere hyperbolische Funktionen (hyperbolic $m = -1$, rotating)

Wie bei den trigonometrischen Funktionen können wegen

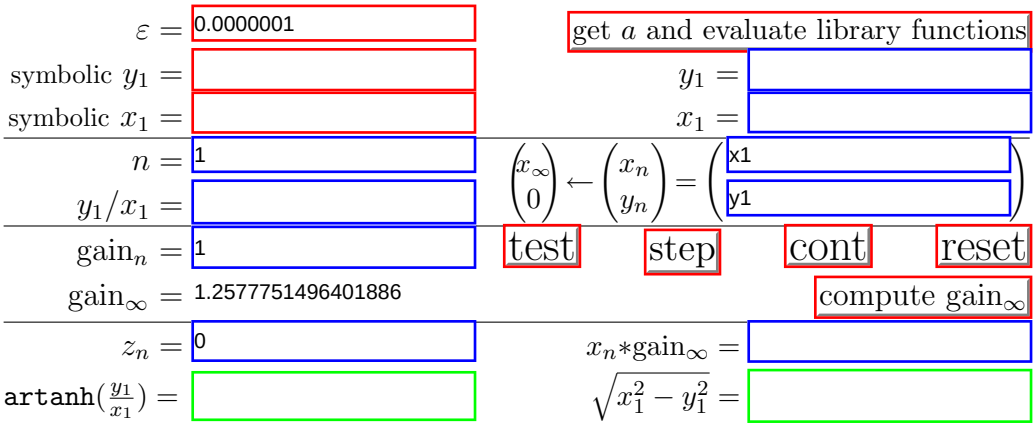
$$\tanh x = \frac{\sinh x}{\cosh x} \quad \text{and} \quad \coth x = \frac{\cosh x}{\sinh x} \quad \text{and} \quad \exp x = \sinh x + \cosh x$$

die hyperbolischen Funktionen tanh und coth sowie die Exponential-Funktion leicht mit CORDIC-Algorithmen ausgewertet werden.

7.4 Inverse hyperbolische Funktionen (hyperbolic $m = -1$, vectoring)

CORDIC-Algorithmen berechnen die inversen hyperbolischen Funktionen analog den trigonometrischen Funktionen und ihrer Inversen.

Die lokal verfügbare Funktion `artanh` und die JavaScript-Bibliotheksfunktion `Math.sqrt` dienen als Referenz (in grün).



Wegen $\operatorname{arcoth}(x) = \operatorname{artanh}(\frac{1}{x})$ sowie

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

berechnet der CORDIC-Algorithmus für artanh ebenso die anderen inversen hyperbolischen Funktionen.

Üb.: Welche Argumente sind im obigen Algorithmus nicht zulässig? Wie ist mit diesen unzulässigen Argumenten zu verfahren?

7.4.1 Logarithmus und Quadrat-Wurzel (hyperbolic $m = -1$, vectoring)

Wegen $\tanh \ln \sqrt{x} = \frac{e^{\ln\sqrt{x}} - e^{-\ln\sqrt{x}}}{e^{\ln\sqrt{x}} + e^{-\ln\sqrt{x}}} = \frac{\sqrt{x}-1/\sqrt{x}}{\sqrt{x}+1/\sqrt{x}} = \frac{x-1}{x+1}$ und $\ln x = 2 \operatorname{artanh} \frac{x-1}{x+1}$ gibt es auch einen CORDIC-Algorithmus zur Berechnung des Logarithmus.

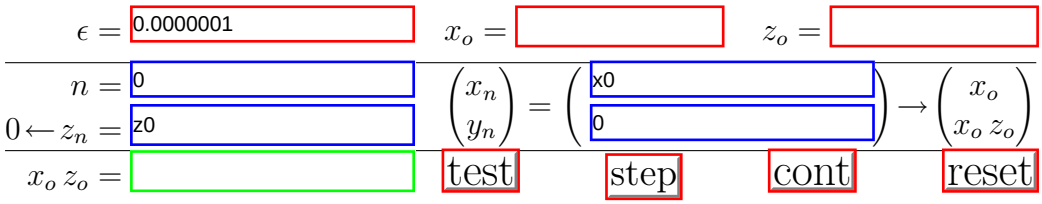
Um $\sqrt{r} = \sqrt{(r+\frac{1}{4})^2 - (r-\frac{1}{4})^2}$ zu berechnen, sei $x_o = r + \frac{1}{4}$, $y_o = r - \frac{1}{4}$ und

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

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

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

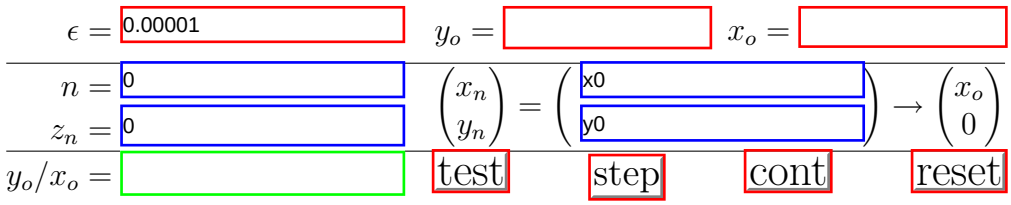
Es gibt eine CORDIC-Version der (Fest- oder Gleitkomma) Multiplikation.



Üb.: Wie ist mit dem beschränkten Argument-Bereich umzugehen?

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

Es gibt eine CORDIC-Version der (Fest- oder Gleitkomma) Division.



Üb.: Wie mit dem beschränkten Argument-Bereich umgehen?

7.7 CORDIC-Algorithmen vereinigt

CORDIC-Algorithmen der beiden Typen drehend (*rotating*) und vektorierend (*vectoring*) in den drei Modi linear ($m = 0$), zirkulär ($m = 1$) und hyperbolisch ($m = -1$) lassen sich in einheitlicher Weise darstellen und daher auch gemeinsam effizient implementieren. Sei T_k dabei die Matrix

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

Die folgende Tabelle gibt dann die Details wieder.

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

7.8 Konvergenz der CORDIC Algorithmen

Die Konvergenz der CORDIC Algorithmen beruht auf folgendem Satz.

Satz
Sei $e_o \geq e_1 \geq e_2 \geq \ldots \geq e_n > 0$ eine endliche Folge reeller Zahlen mit

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

Sei r reell mit $|r| \leq \sum_{j=0}^{n} e_j$. Sei die Folge s_k induktiv definiert durch $s_o = 0$ und $s_{k+1} = s_k + e_k \operatorname{sgn}(r - s_k)$ für $k \in \mathbb{N}_o$. Dann gilt

$$|r - s_k| \leq e_n + \sum_{j=k}^{n} e_j \quad \text{und speziell} \quad |r - s_{n+1}| \leq e_n. \quad \text{d.h.} \quad \lim_{k\to\infty} s_k = r$$

Beweis durch vollständige Induktion:
vorausgesetzt $|r - s_k| \leq e_n + \sum_{j=k}^{n} e_j$. Um $|r - s_{k+1}| \leq e_n + \sum_{j=k+1}^{n} e_j$ zu zeigen, unterscheide die vier Fälle:

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

- $\boldsymbol{r - s_k - e_k > 0}$ Dann $|r - s_{k+1}| = r - s_k - e_k \leq e_n + \sum_{j=k}^{n} e_j - e_k = e_n + \sum_{j=k+1}^{n} e_j$.
- $\boldsymbol{r - s_k - e_k < 0}$ d.h. $s_k < r < s_k + e_k$ oder $0 < r - s_k < e_k$ oder $-e_k < s_k - r < 0$ so daß $|r - s_{k+1}| = |r - s_k - e_k| = s_k - r + e_k \leq e_k \leq e_n + \sum_{j=k+1}^{n} e_j$.

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

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

qed

Folgerung
Die e_k der CORDIC-Algorithmen der drei Modi erfüllen die Voraussetzungen des vorstehenden Satzes. Damit sind die CORDIC-Algorithmen konvergent.

Beweis

$\boldsymbol{m = 0}$ Dann ist $e_k = 2^{-k}$ mit $e_o \geq e_1 \geq \ldots \geq e_n > 0$ und es folgt $e_k = 2^{-k} \leq e_n + \sum_{j=k+1}^{n} e_j$, weil für die rechte Seite $e_n + \sum_{j=k+1}^{n} e_j = 2^{-n} + \sum_{j=k+1}^{n} 2^{-j} = 2^{-n} + 2^{-(k+1)} \sum_{i=0}^{n-k-1} 2^{-i} = 2^{-n} + 2^{-(k+1)} \frac{1-2^{-(n-k)}}{1-2^{-1}} = 2^{-n} + 2^{-(k+1)}\, 2(1 - 2^{-(n-k)}) = 2^{-n} + 2^{-k}(1 - 2^{-(n-k)}) = 2^{-k} = e_k$ gilt.

$\boldsymbol{m = 1}$ Dann ist $e_k = \arctan 2^{-k}$ mit $e_o \geq e_1 \geq \ldots \geq e_n > 0$ wegen der Monotonie des Arcus Tangens. Wegen $2 \arctan x = \arctan \frac{2x}{1-x^2}$ und der Monotonie gilt $\arctan x \leq 2 \arctan \frac{x}{2} = \arctan \frac{2\,x/2}{1-x^2/4}$, d.h.

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

und durch wiederholte Anwendung

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

$\boldsymbol{m = -1}$ Dann $e_k = \operatorname{artanh} 2^{-k} = \frac{1}{2} \ln \frac{1+2^{-k}}{1-2^{-k}}$. Um Konvergenz sicherzustellen, werden alle hyperbolischen Rotationen außer der ersten dupliziert. Daher ist

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

zu zeigen, nämlich durch Induktion in $n - k$, d.h.

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

Im Fall $k = 1$ ist $e_{n-1} \leq e_n + 2\, e_n = 3\, e_n$ zu zeigen, äquivalent zu

$$0 \leq 3\, e_n - e_{n-1} = \frac{1}{2} \ln \left(\left(\frac{1+2^{-n}}{1-2^{-n}} \right)^3 \frac{1-2^{-n+1}}{1+2^{-n+1}} \right)$$

was aufgrund der Monotonie des Logarithmus äquivalent ist zu

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

und damit richtig für $n > 1$.
Unter der Voraussetzung $e_{n-k} \leq e_n + 2 \sum_{j=n-k+1}^{n} e_j$ ist die Induktionsbehauptung $e_{n-k-1} \leq e_n + 2 \sum_{j=n-k}^{n} e_j$ äquivalent zu $e_{n-k-1} \geq 3\, e_{n-k}$, was schon im Induktionsanfang verifiziert wurde.

qed

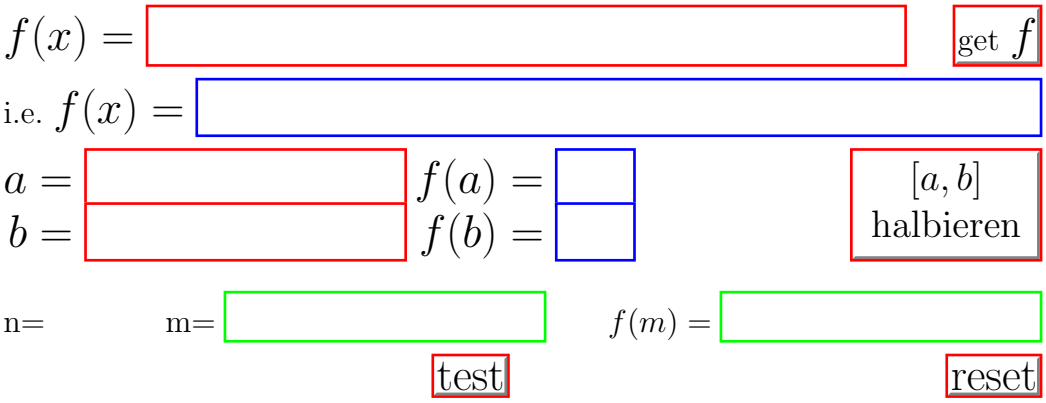
8 Bestimmung von Nullstellen

Die numerische Bestimmung von Nullstellen ist immer dann unverzichtbar, wenn beispielsweise zur Extremwert-Bestimmung die Nullstellen der ersten Ableitung nicht analytisch bestimmt werden können.

Dies gilt schon für alle Polynome höherer Ordnung.

8.1 Nullstellenbestimmung per Intervallhalbierung

Das Feld für die Funktionswerte ist bewußt 'zu klein' gewählt, da bei diesem Verfahren nur deren Vorzeichen interessieren: das Start-Intervall $[a, b]$ mit der Invarianten $f(a)f(b) < 0$ wird durch $m = (a + b)/2$ halbiert und durch das rechte oder linke Teil-Intervall ersetzt, das die Invariante erfüllt.



Üb.: Abbruch-Kriterien? Start-Intervall?

Üb.: Berechne Extrema von Funktionen.

8.2 Nullstellenbestimmung per regula falsi

Das Start-Intervall $[a, b]$ mit der Invarianten $f(a)f(b) < 0$ wird durch die Nullstelle $m = a - f(a)\frac{b-a}{f(b)-f(a)}$ der Sekanten geteilt und durch das rechte oder linke Teil-Intervall ersetzt, das die Invariante erfüllt.

$f(x) =$ [ ] get f

i.e. $f(x) =$ [ ]

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

$b =$ [ ] $f(b) =$ [ ]

n= $m =$ [ ] $f(m) =$ [ ]

test reset

Üb.: Abbruch-Kriterien? Vergleich zur Intervall-Halbierung?

Üb.: Berechne Extrema von Funktionen.

8.3 Nullstellenbestimmung per Newton Algorithmus

Es wird unterstellt, daß die Bedingungen für die Konvergenz des Newton-Verfahrens erfüllt sind: die gesuchte Nullstelle wird durch die Nullstelle $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ der Tangenten in $(x_n, f(x_n))$ approximiert.

$f(x) =$ [ ] 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

Üb.: Berechne $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$, π, $\pi/4$ u.ä.

Üb.: Berechne Extrema von Funktionen.

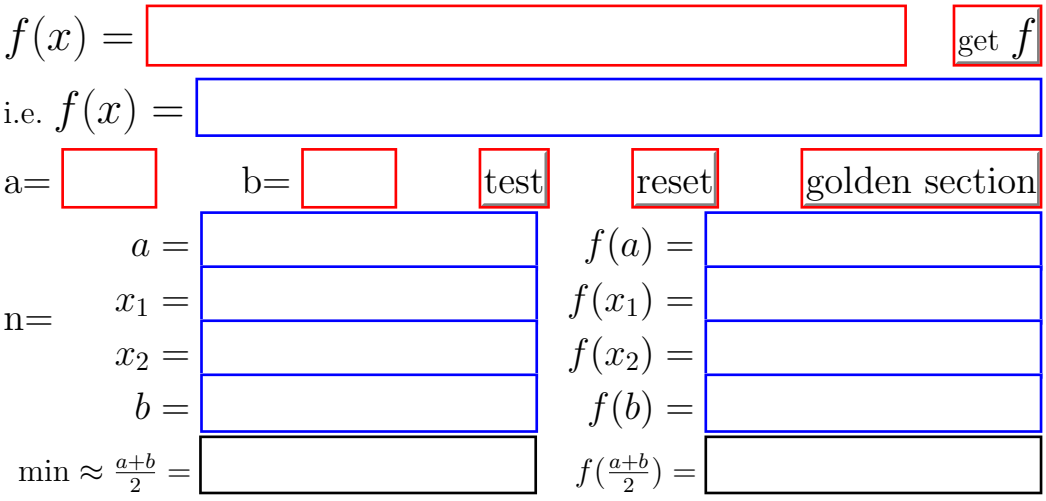
Üb.: Abbruch-Kriterien? Vergleich zur regula falsi?

9 Optimierung

Numerische Optimierung bedeutet, Minima einer Funktion $f(x)$ zu bestimmen. Die Maxima von f sind die Minima von $-f$.

9.1 Optimierung mit dem Goldenen Schnitt

Wie bei der Intervall-Halbierung muß auch hier bekannt sein, daß f in $[a, b]$ genau ein Minimum x^* besitzt. f ist dann links von x^* monoton fallend und rechts von x^* monon wachsend. Wenn $f(x_1) < f(x_2)$ für $a < x_1 < x_2 < b$, dann $x^* \notin (x_2, b]$ und daher $x^* \in [a, x_2)$, und wenn $f(x_1) > f(x_2)$ für $a < x_1 < x_2 < b$, dann $x^* \notin [a, x_1)$ und daher $x^* \in (x_1, b]$. So wird die Länge des Intervalls, das x^* enthält, stetig verkürzt. Am besten ist es, wenn in jedem Schritt die Funktion nur einmal neu ausgewertet werden muß und wenn die Position von x_1 und x_2 relativ zu a and b konstant und invariant ist, nämlich $x_1 = a + (1 - \tau)(b - a)$ und $x_2 = a + \tau(b - a)$ für $\tau = (\sqrt{5} - 1)/2$.

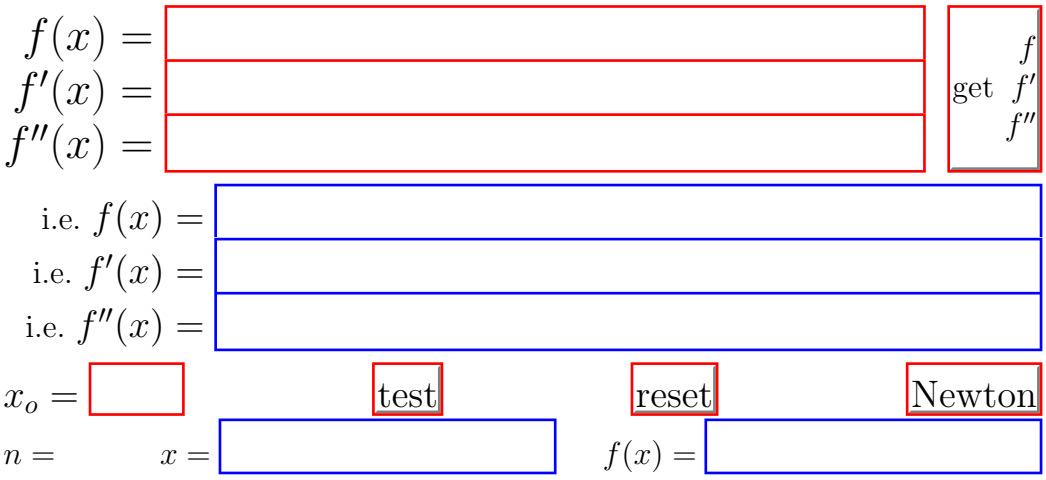


Üb.: Berechne Minima von Funktionen wie $f(x) = e^{-x} \sin x$ oder $g(x) = (x - 1)^n$ usw.

Üb.: Abbruch-Kriterien?

9.2 Optimization per Newton

Warum nicht einfach die Nullstellen der ersten Ableitung mit dem Newton-Verfahren bestimmen? Natürlich brauchen wir dann erste und zweite Ableitung der zu minimierenden Funktion.



Üb.: Berechne Minima von Funktionen wie $f(x) = e^{-x} \sin x$ oder $g(x) = (x-1)^n$ usw.

Üb.: Abbruch-Kriterien?

Üb.: Wenn man nach den Nullstellen der ewrsten Ableitungen sucht, findet man nicht notwendig Minima. Finde Beispiele für dieses Phänomen.

10 Integration

Numerische Integration für $\int_a^b f(x)\,dx$ ist Notnagel immer dann, wenn keine geschlossene Stammfunktion angegeben werden kann.

Zur Bestimmung der Komplexität der Verfahren wird die Anzahl der Berechnungen von Funktionswerten verwendet.

10.1 Integration per Trapez-Regel

Das Integral wird duch die Summe der Flächeninhalte von Trapezen approximiert.

$f(x) =$ [ ] 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

Üb.: Berechne bekannte Integrale wie etwa $\int_0^\pi \sin x\,dx$ oder $\int_1^2 x^{-2}\,dx$ usw.

Üb.: Abbruch-Kriterien? Optimierung durch Verdoppelung statt Inkrementierung von n ?

10.2 Integration per Simpson-Regel

Die Funktion wird stückweise durch Parabeln approximiert. Die Summe der Integrale dieser Parabeln approximiert dann das gesuchte Integral.

n (geradzahlig) wird jeweils um 2 inkrementiert.

$f(x) =$ [ ] 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

Üb.: Berechne bestimmte Integrale von Polynomen zweiten Grades. Warum ist die Approximation hier immer korrekt?

Üb.: Berechne Integrale wie $\int_1^e \frac{1}{x}\,dx$ usw.

Üb.: Vergleich zur Trapez-Regel? Verallgemeinerungen?

11 Gewöhnliche Differentialgleichungen erster Ordnung

Die Verfahren von Euler, Heun, Euler-Cauchy und Runge-Kutta illustrieren, wie gewöhnliche Differentialgleichungen erster Ordnung numerisch gelöst werden können, insbesondere wenn keine geschlossenen Lösung bekannt ist.

Dies sei zunächst für Anfangswert-Probleme erster Ordnung erläutert.

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

Grundsätzlich sind also die Funktion $f(x, y)$, x_o und $y_o = y(x_o)$ als Anfangsbedingung sowie die Anzahl n der Schritte einzugeben, um den gesuchten Funktionswert $y_n \approx y(x_n)$ an der (Ziel-) Stelle x_n approximieren zu können. Dann können die Ergebnisse der verschiedenen Verfahren verglichen werden. Die Qualität der Verfahren zeigt sich erst im Vergleich mit der exakten Lösung $y(x)$, falls diese verfügbar ist.
Der Test-Button erspart die Eingabe der Differentialgleichung $y' = y$ mit der Anfangsbedingung $y(0) = 1$ und der Lösung $y(x) = e^x$.

Vorgestellt werden einige klassiche Verfahren zur numerischen Lösung von Anfangswertproblemen, nämlich die Verfahren von Euler, Heun und Euler-Cauchy sowie das klassiche Runge-Kutta-Verfahren.

11.1 Euler-Verfahren

Approximation durch die Tangente in der jeweils letzten Stützstelle liefert das Euler-Verfahren.

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

11.2 Heun-Verfahren

Das Heun Verfahren ist ein Prediktor-Korrektor Verfahren: y_{n+1} wird vermittels der Sekanten durch (x_n, y_n) berechnet, deren Steigung gerade der Mittelwert (Korrektor) der Steigung in x_n und der Steigung $f(x_{n+1}, \bar{y}_{n+1})$ in

$(x_{n+1}, \bar{y}_{n+1})$ (Prädiktor) ist.

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

11.3 Euler-Cauchy-Verfahren

Das Euler-Cauchy-Verfahren approximiert die Tangenten-Steigung im Mittelpunkt $x_{n+1/2} = \frac{1}{2}(x_n + x_{n+1})$ von $[x_n, x_{n+1}]$,
also $y(x_{n+1/2}) \approx y_{n+1/2} = y_n + \frac{h}{2} f(x_n, y_n)$.

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

11.4 Runge-Kutta-Verfahren

Das klassische Runge-Kutta Verfahren approximiert die gesuchte Funktion durch eine Sekante, deren Steigung sich als gewichetete Summe der vier Steigungen y'_i in x_i, $\bar{y}'_{i+1/2}$ und $\bar{\bar{y}}'_{i+1/2}$ in $x_{i+1/2}$ sowie $\bar{y}'_{i+1}$ in x_{i+1} ergibt.

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

11.5 interaktive Synopse dieser Verfahren

$f(x,y) =$ [ ] 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 $y =$	
		exakt $y(x) = y($ [ ] $) =$	

test | ein Schritt | repeat | reset

Üb.: Experimentiere mit $y' = \cos x$ und $y(0) = 0$ oder $y' = xy$ und $y(0) = 1$ o.ä.

Üb.: Vergleiche die verschiedenen Verfahren anhand ihrer Komplexität, d.h. der Anzahl der Funktionsaufrufe von $f(x,y)$.

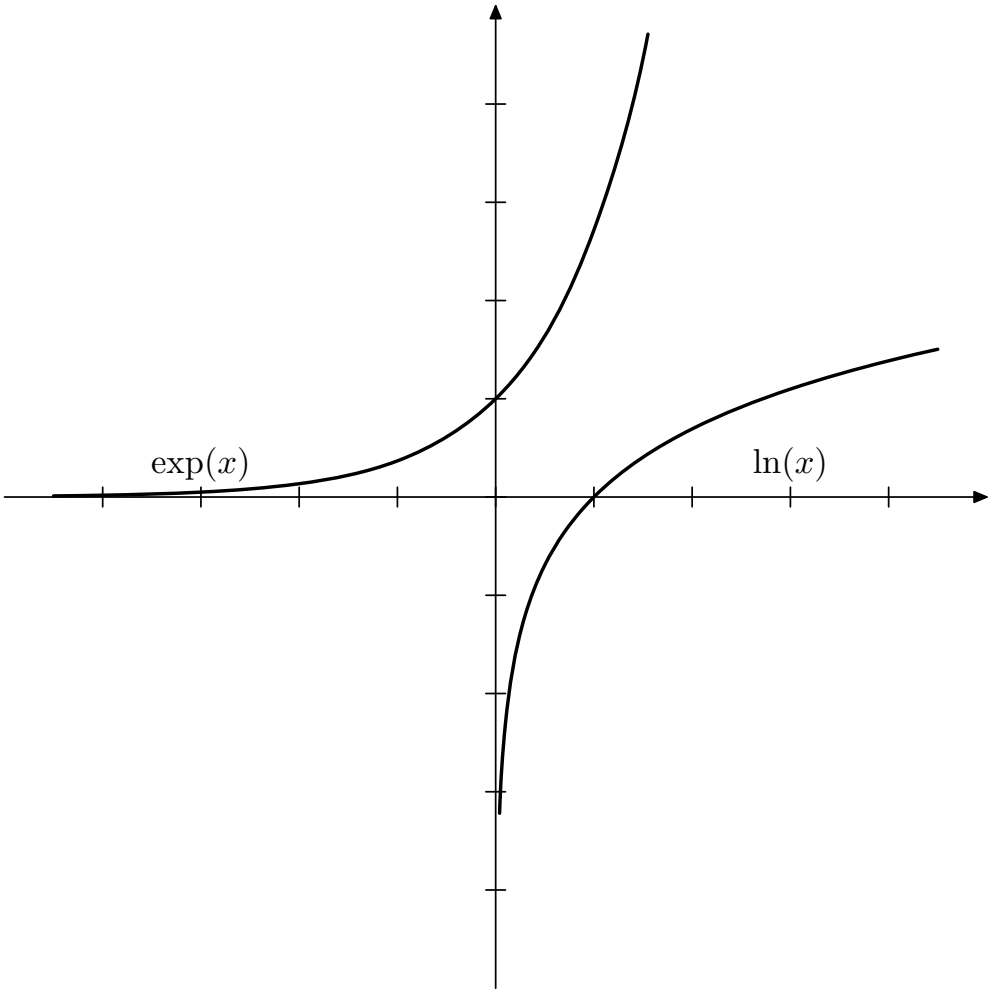
11.6 Systeme gewöhnlicher Differentialgleichungen erster Ordnung

Die Verfahren aus den vorangehenden Abschnitten können auch zur Lösung von Systemen gewöhnlicher Differentialgleichungen erster Ordnung eingesetzt werden. Als default Beispiel dient die Entwicklung von Jäger- und Beute-Populationen über die Zeit.

nzi

A Graphen elementarer Funktionen

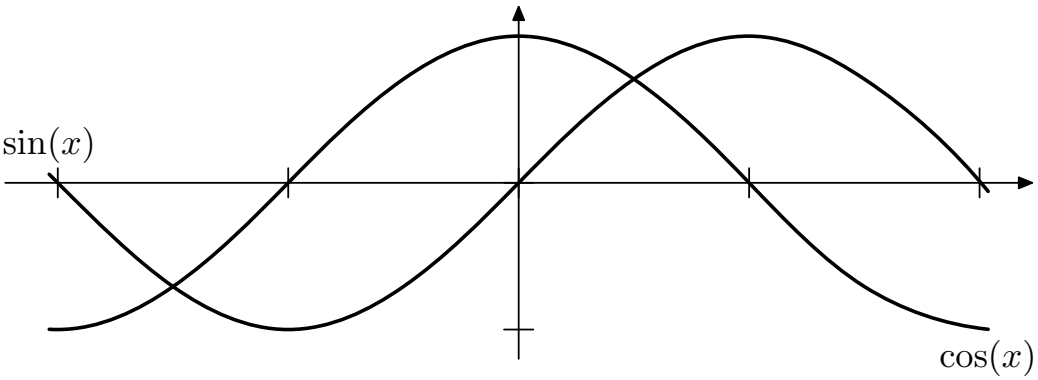
A.1 Exponential-Funktion und Logarithmus



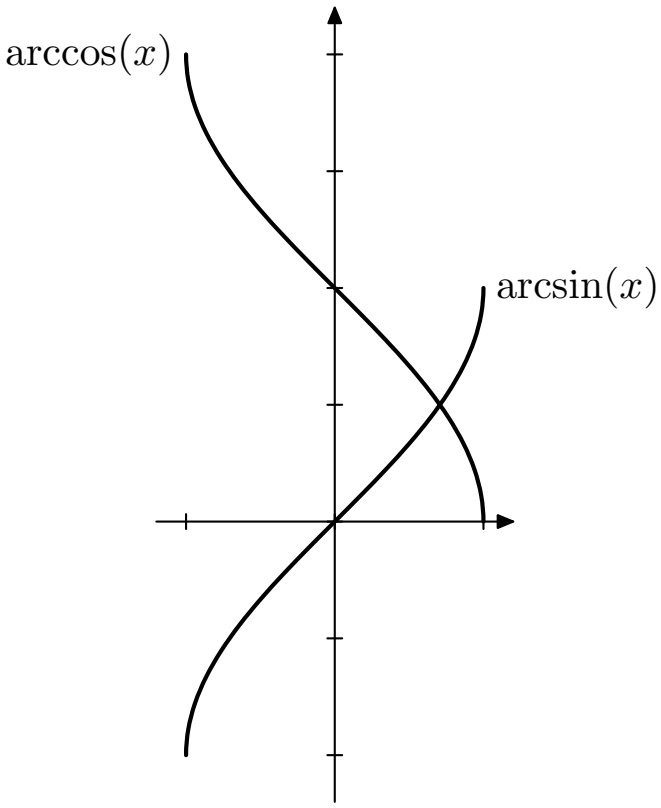
exp ln

Üb.: Bestimme Längeneinheiten und Eigenschaften der Funktionen.

A.2 Sinus und Cosinus und deren Inverse

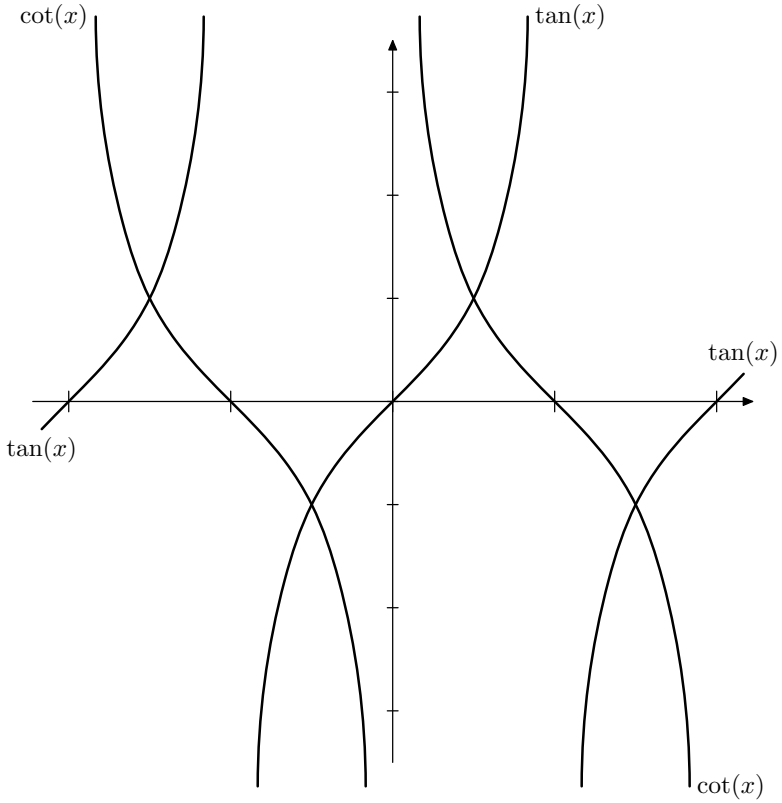


sin cos

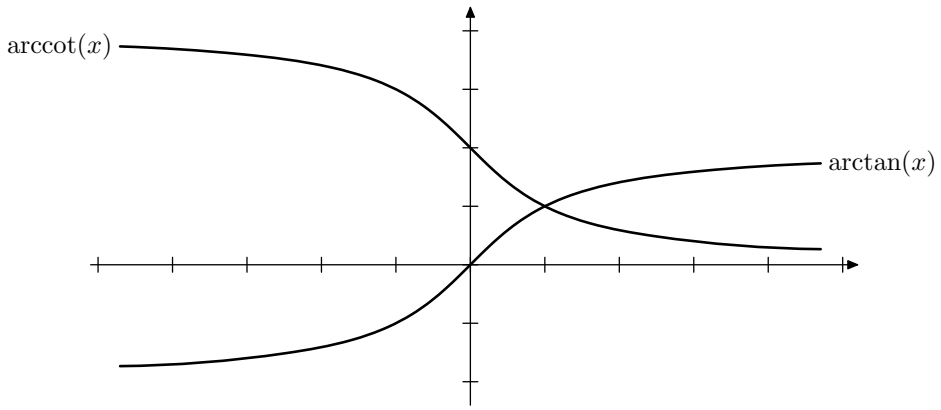


Üb.: Bestimme Längeneinheiten und Eigenschaften der Funktionen.

A.3 Tangens und Cotangens und deren Inverse

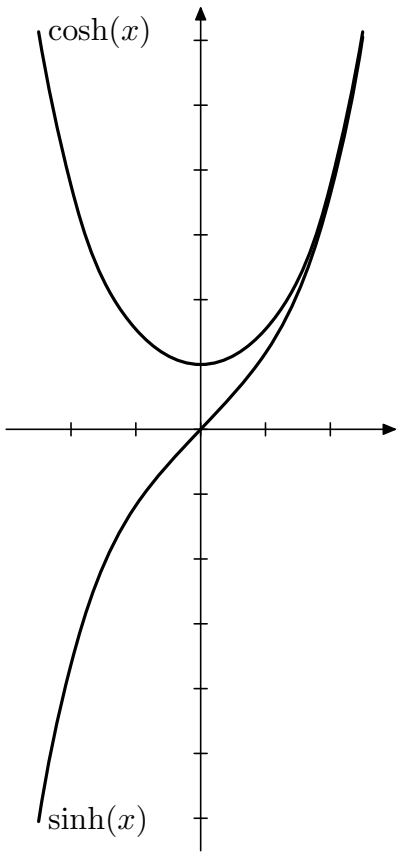


tan cot

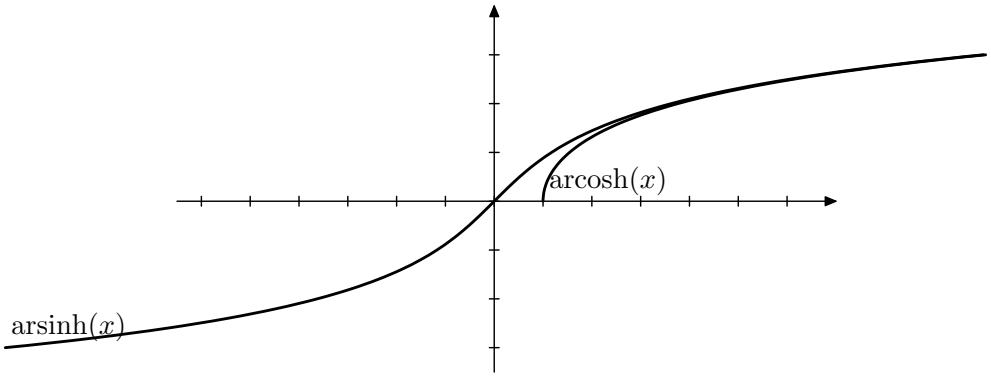


Üb.: Bestimme Längeneinheiten und Eigenschaften der Funktionen.

A.4 Sinus und Cosinus hyperbolicus und deren Inverse

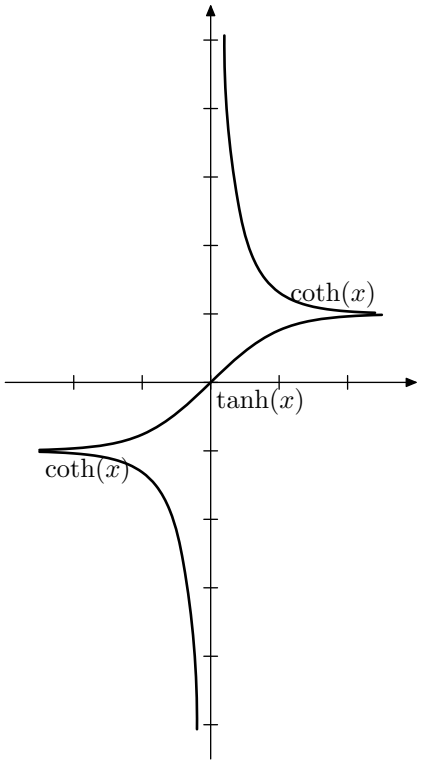


sinh cosh

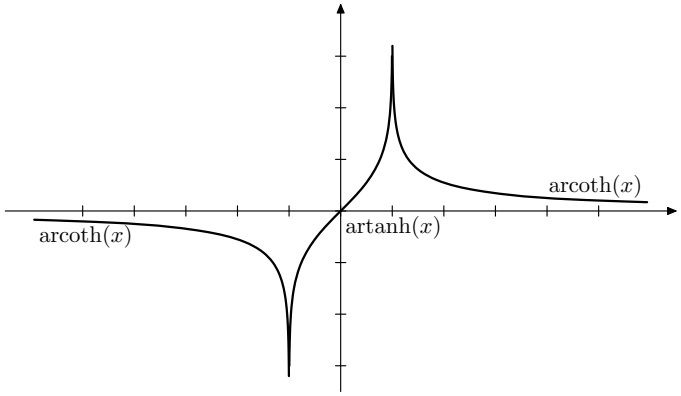


Üb.: Bestimme Längeneinheiten und Eigenschaften der Funktionen.

A.5 Tangens hyperbolicus und Cotangens hyperbolicus und deren Inverse



tanh coth



Üb.: Bestimme Längeneinheiten und Eigenschaften der Funktionen.

Inhaltsverzeichnis