Autodiff
TU Ilmenau

Lukas Elbert (47100) Sebastian Semper (46382)

Juli 2011

Inhaltsverzeichnis

1 Aufgabenstellung

Entwickeln Sie in C++ eine Klasse Autodiff, die mittels automatischer Differentiation die 1. Ableitung einer vorgegebenen Funktion berechnet. Implementieren Sie die Grundregeln der Differentiationsarithmetik und wenden Sie diese auf reelle und transzendente Funktionen an. Es sind zudem geeignete Konstruktoren und Destruktoren bereitzustellen.

Überladen Sie die Operatoren +,-,*,/, den Zuweisungsoperator =, die Ein- und Ausgabeoperatoren sowie mathematische Standartfunktionen. Entwickeln Sie ein Demonstrationsprogramm mit einer einfachen Menüsteuerung, welches die Anwendung der Klassen Autodiff anhand selbsgewählter Beispiele zeigt.

Legen Sie zudem kurz die theoretischen Sachverhalte Formeln usw. dar.
Geben Sie einige Beispiele an und fügen sie den durchkommenteierten Quelltext des Programmes an.

2 Theoretische Grundlagen

2.1 Differentiationsarithmetik

Als Differentiationsarithmetik ist eine Arithmetik geordneter Paare der Form

$$U = (u, u') \text{ mit } u, u' \in \mathbb{R}$$

beschrieben. Dabei beinhaltet die erste Komponente von U den Wert $u(x)$ der Funktion $u : \mathbb{R} \to \mathbb{R}$ am Punkt $x \in \mathbb{R}$. Die zweite Komponente beinhaltet den Wert der ersten Ableitung $u'(x)$ an der Stelle $x \in \mathbb{R}$.

Für unsere Differentiationsarithmetik geordneter Paare gelten dabei die folgenden Grundregeln:
(i) $U + V = (u, u') + (v, v') = (u + v, u' + v')$
(ii) $U - V = (u, u') - (v, v') = (u - v, u' - v')$
(iii) $U * V = (u, u') * (v, v') = (u * v, u' * v')$
(iv) $U/V = (u, u')/(v, v') = (u/v, (u' - u * v'/v)/v)$

Zudem gelten für die unabhängige Variable x und die beliebige Konstante c:
(i) $X = (x, 1)$ wegen $\frac{dx}{dx} = 1$
(ii) $C = (c, 0)$ wegen $\frac{dc}{dx} = 0$

Mit den obigen sechs Regeln lässt sich nun unsere Differentiationsarithmetik auf reelle Funktionen anwenden. Dazu ersetzt man die unabhängige Variable $x \in \mathbb{R}$ in einem Ausdruck $f(x)$ durch $x = (x, 1)$ und alle Konstanten $c \in \mathbb{R}$ durch $C = (c, 0)$. Damit liefert uns die Berechnung von f mit den genannten Regeln der Differentiationsarithmetik das geordnete Paar

$$Y = f(X) = f((x, 1)) = (f(x), f'(x)).$$

Somit können wir also in unserem C++ Programm diese Regeln für die Differentiationsarithmetik anwenden.

Da es die Aufgabenstellung fordert, dass unser Autodiff-Programm auch für transzendente Funktionen arbeitet, benötigen wir zudem natürlich noch die Kettenregel, die für geordnetete Paare wie folgt für die gegebene Variable $U = (u(x), u'(x))$ definiert ist:

$$f(U(x)) = f((u(x), u'(x))) = (f(u(x)), \frac{df}{du} \cdot u'(x))$$

Für die transzendenten Funktionen sind die Ableitungen wie folgt definitert:

- $f(x) = \sin(g(x))$, $f'(x) = g'(x)\cos(g(x))$
- $h(x) = \cos(g(x))$, $h'(x) = g'(x)(-\sin(g(x))$
- $i(x) = \ln(g(x))$, $i'(x) = g'(x) \cdot \frac{1}{g(x)}$
- $j(x) = \exp(g(x))$, $j'(x) = g'(x) \cdot \exp(g(x))$

Hier ist schön zu sehen, dass wir die Kettenregel für die transzendenten Funktionen brauchen.

Zusätzlich zu diesen Funktionen haben wir noch weitere in unserem Programm hinzugefügt, sie werdem über die eben beschrieben Funktionen definiert:

- $\tan(x) = \frac{\sin(x)}{\cos(x)}$
- $\cot(x) = \frac{\cos x}{\sin(x)}$
- $\sinh(x) = \frac{1}{2} \cdot (e^x - e^{-x})$
- $\cosh(x) = \frac{1}{2} \cdot (e^x + e^{-x})$
- $\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$
- $\coth(x) = \frac{\cosh(x)}{\sinh(x)}$

Für sie sind keine weiteren besonderen Ableitungsregeln explizit zu definieren, da sie über die anderen, klar bestimmten, transzendenten Funktionen definiert sind. Dies funktioniert, da wir im Programm später die Grundrechenarten und die grundlegenden transzendenten Funktionen bereits für `autodiff` überladen haben werden. Zum Ableiten benutzt man die Produkt- und Quotientenregel (siehe Regeln weiter oben).

2.2 Zweite Ableitungen

Zur ersten Ableitung haben wir in unsere Klasse um die zweite Ableitung erweitert. Hierfür gibt es wiederum weitere Grundregeln, die es zu beachten gilt.

Dabei betrachten wir das Tripel $U = (u, u', u'')$ mit $u, u', u'' \in \mathbb{R}$. Hier beschreibt u den Funktionswert, u' den Wert der ersten Ableitung und u'' den Wert der zweiten Ableitung.

Die Regeln für die zweite Ableitung sind also wie folgt

(i)Konstante: $u(x) = c$, c-Kosntante, $U = (c, 0, 0)$
(ii) Differnetiationsvariable x: $u(x) = x$ $U = (x, 1, 0)$
(iii) Arithmetik: $W = U \circ V$ mit $\circ \in \{+, -, *, /\}$ für $W = (w, w', w''), U = (u, u', u'')$, $V = (v, v', v'')$
(a) Regeln für w und w' sind oben bereits für die erste Ableitung definiert.
(b) Regeln für w'':
$W = U + V \Rightarrow w'' = u'' + v''$
$W = U - V \Rightarrow w'' = u'' - v''$
$W = U * V \Rightarrow w'' = uv'' + 2u'v' + vu''$
$W = U/V \Rightarrow w'' = (u'' - 2w'v' - wv'')/v, v \neq 0$

Für $U = (u, u', u'')$ liefert uns zudem die Kettenregel für $W = f(U) = f((u, u', u'')) :$
$w = f(u)$
$w' = f'(u) \cdot u'$
$w'' = f'(u) \cdot u'' + f''(u) \cdot (u')^2$, d.h.

$$W = f(U) = (f(u), f'(u) \cdot u', f'(u) \cdot u'' + f''(u) \cdot (u')^2).$$

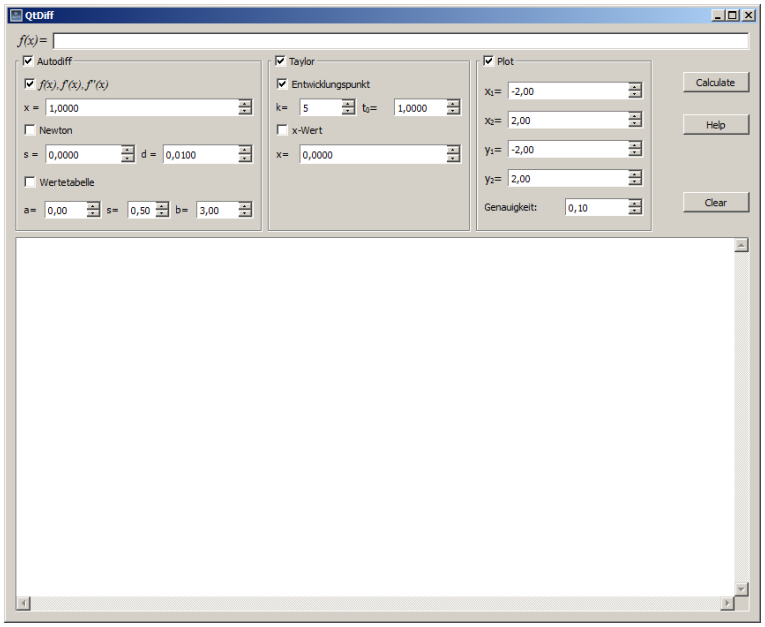
Dabei sind f' und f'' die gegebenen ersten und zweiten Ableitungen von f.

Auch hier benutzen wir die transzendenten Funktionen, deren Ableitungen wie folgt definiert und benutzt werden:
Für die transzendenten Funktionen sind die Ableitungen wie folgt definitert:

- $f(x) = \sin(g(x))$, $f''(x) = (g'(x))^2 \cdot (-\sin(g(x))) + g''(x) \cdot \cos(g(x))$
- $h(x) = \cos(g(x))$, $h''(x) = (g'(x))^2 \cdot (-\cos(g(x))) + g''(x) \cdot (-\sin(g(x)))$
- $i(x) = \ln(g(x))$, $i''(x) = (g'(x))^2 \cdot (-\frac{1}{(g(x))^2}) + g''(x) \cdot \frac{1}{g(x)}$
- $j(x) = \exp(g(x))$, $j''(x) = (g'(x))^2 \cdot \exp(g(x)) + g''(x) \exp(g(x))$

Unsere weiteren transzendenten Funktion sind wiederum (wie bei der ersten Ableitung), über diese Funktionen definiert.

3 Gui-Oberfläche



Oberfläche unter Winows 7

Wir haben uns entschieden eine optisch ansprechendere Oberfläche für unser Programm zu entwickeln, die unserer Meinung nach auch wesentlich benutzerfreundlicher ist.
Dazu benutzten wir das freie Gui-Framework Qt, welches unter der GNU-Lizenz verteilt wird und somit frei nutzbar ist. Der Vorteil dieses Frameworks ist, dass es Plattform unabhängig ist und sich unser Programm optisch in jedes Betriebssystem sehr gut integriert.
Am wichtigsten ist die Eingabezeile am oberen Rand des Fensters. Hier kann der Nutzer einen beliebigen Funktionsterm eingeben, welcher dann vom Programm verarbeitet wird.
Man kann in der Sektion Autodiff zwischen drei verschiedenen Funktionen wählen: Zum einen gibt es die Möglichkeit sich an einem bestimmten selbstwählbaren Punkt den Funktionswert, sowie die Werte der ersten und zweiten Ableitung ausgeben zulassen. Zudem gibt es den Unterpunkt Newton, bei dem man eine Newton-Iteration ausführen lassen kann und sich so über den Startwert s und die Genauigkeit d die Nullstelle annähern kann. Zuletzt haben wir die Auswahl einer Wertetabelle zur Verfügung gestellt, über die sich der Nutzer, in einem selbstgewählten Intervall mit bestimmbaren Schritten Werte ausgeben lassen kann.
Wieterhin existiert sowohl für `autodiff` als auch für `taylor` die Möglichkeit die Ergebniss auch Grafik darstellen zu lassen. Bei `autodiff` werden die Funktion selbst und ihre beiden Ableitungen geplottet. Bei `taylor` werden alle Taylor-Polynome bis zum in der Rubrik Taylor ausgewählten k geplottet. Dabei kann man sowohl das x- als auch das y-Achsen Intervall bestimmen. Zudem kann man die Genauigkeit des Plots bestimmen.
Wenn der Nutzer seine gewünschte Funktion eingegeben hat, kann er sich das Ergebnis entweder über den Button „Calculate“ oder die Eingabtetaste ausgeben lassen.
Um das Ausgabefeld von den bereits ausgerechneten Ergebnissen zu bereinigen haben wir den Button „Clear“ eingebaut.

Unter dem „Help“-Button haben wir die Befehle beschrieben, die in unserem Programm zur Verfügung stehen. Hier bekommt man einen Überblick über die möglichen Funktionen. Weiterhin haben wir hier eine kleine Einführung in unser Programm gegeben.

4 Programmstruktur und Nutzungshinweise

4.1 Grundstruktur

Die grafische Oberfläche dient einerseits dazu um die Dateneingabe zu ermöglichen und andererseits ist sie zuständig für die Ausgabe der errechneten Ergebnisse.
Mit der vom Benutzer gemachten Eingabe wird eine Instanz der Klasse parser erstellt. Deren Konstruktor wird die Eingabe als String übergeben. Dieser Konstruktor evaluiert den eingegebenen Ausdruck. Dabei wird geprüft, ob die Eingabe einen validen Ausdruck enthält. Wenn dies nicht der Fall ist, wird dies dem Benutzer mitgeteilt und keine Berechnung vorgenommen. Im Falle eines validen Ausdrucks wird aus dem Ausdruck rekursiv ein binärer Baum erstellt. Dabei sind die Trennstellen immer die Operatoren im jeweils vorliegenden Ausdruck. Die Rekursion kommt dadurch zum Ausdruck, dass der String an einem Operator getrennt wird, und auf diese beiden Teilstrings wiederum der Parser angewendet wird.
Sobald der Parser seine Arbeit beendet hat, steht die Instanz von parser bereit um für die Berechnungen von autodiff und taylor verwendet zu werden. Dazu wurden in der Klasse parser für autodiff und taylor zwei rekursive Methoden implementiert, welche dann jeweils mit Differentiations- oder Taylorarithmetik ein bestimmtes Resultat berechnen.
In Abhängigkeit von den ausgewählten Funktionen in der GUI werden die von autodiff und taylor bereitgestellten Funktionen verwendet.

4.2 Programmteile

In `main.cpp` wird lediglich eine Instanz der Klasse gui erstellt, welche alle Informationen und Funktionalität der grafischen Oberfläche enthält. Dadurch wird diese in das Hauptprogramm eingebunden. Alle Funktionen, die in der GUI ausführen, sind Member-Funktionen der Klasse GUI.
In `gui.cpp` wird das Aussehen und die Funktionalität der grafischen Oberfläche implementiert. Diese ist sehr ausführlich kommentiert und erklärt sich größtenteils selbst.
In `autodiff.h` wird unsere Klasse lediglich deklariert. Dabei wird keine Funktion inline definiert, sondern alle werden in `autodiff.cpp` ausgelagert.
In `parser.h` wird eine Klasse deklariert, welche dann in `parser.cpp` definiert wird und dafür zuständig ist, den in der GUI eingegebenen String auszuwerten, um dann den Klassen `taylor` und `autodiff` eine Berechnung zu ermöglichen. Auf die Funktionalität, die von dieser Klasse bereitgestellt wird, wird in `parser.cpp` und `gui.cpp` zurückgegriffen. Diese Dateien werden im Kapitel Implementierung erläutert.

4.3 Nutzungshinweise

Das Programm bietet zwar die Funktionalität einen Ausdruck zu potenzieren. Und zwar mit dem `^`. Doch dieser funktioniert nicht immer korrekt, weshalb er mit Vorsicht zu benutzen ist.
Weiterhin wird beim Plotten in den Unterordner `/plots` für jeden einzelnen Plot ein Bild

erstellt. Diese Bilder erhalten immer ein Suffix in Form einer Zufallszahl. Somit häufen sie sich mit der Zeit in diesem Ordner an und sollten deshalb von Zeit zu Zeit gelöscht werden. Dies bietet aber die Möglichkeit auch sehr einfach Sicherungen von interessanten Plots zu erstellen.

5 Implementierung

5.1 Deklaration

Hier werden die wichtigsten Auszüge aus unserem Quellcode abgedruckt und ergänzend zu den bereits vorhandenen Kommentaren im Code selbst weiter erläutert.
Die Header-Datei `autodiff.h` enthält die Deklaration der Klasse. Hier wird deklariert, welche Funktionen Freundfunktionen der Klasse sind, welche Datentypen diese als Eingabe benötigen und welche zurückgegeben werden. Weiterhin werden die Deklarationen für die überladenen Operatoren angegeben. Die öffentlichen Funktionen für die Klasse sind größtenteils Schnittstellenfunktionen, um von außen Daten manipulieren und auslesen zu können.

```
/*########
autodiff.h
########*/

#ifndef AUTODIFF_H
#define AUTODIFF_H
#include <iostream>
#include "math.h"
#include "string.h"

using namespace std;

class autodiff{
        friend autodiff pow(autodiff,autodiff);
        friend autodiff exp(autodiff);
        friend autodiff ln(autodiff);
        friend autodiff sin(autodiff);
        friend autodiff cos(autodiff);
        friend autodiff tan(autodiff);
        friend autodiff cot(autodiff);
        friend autodiff cosh(autodiff);
        friend autodiff sinh(autodiff);
        friend autodiff tanh(autodiff);
        friend autodiff coth(autodiff);
        friend autodiff operator *(autodiff, autodiff);
        friend autodiff operator /(autodiff, autodiff);
        friend autodiff operator +(autodiff, autodiff);
        friend autodiff operator -(autodiff, autodiff);
        friend autodiff operator -(autodiff);
        friend autodiff operator *(double, autodiff);
    private:
        //Das Zahlentripel fuer Funktionswert, Wert der Ableitung und Wert
```

```
        der zweiten Ableitung
        double u;
        double v;
        double w;
    public:
        //Schnittstellenfunktionen zur Eingabe
        void set_u(double);
        void set_v(double);
        void set_w(double);

        //Schnittstellenfunktionen zur Ausgabe
        double get_u();
        double get_v();
        double get_w();

        //Methode zur einfachen Ausgabe
        string toStdString();

        //Zuweisungsoperator
        autodiff &operator =(const autodiff &s);

        //Konstruktor
        autodiff(double, double, double);

        //Kopierkonstruktor
        autodiff(const autodiff &a);

        //Destruktor
        ~autodiff();
};
#endif //AUTODIFF_H
```

5.2 Definition

In `autodiff.cpp` wurden alle in `autodiff.h` deklarierten Funktionen vollständig implementiert. Besonders bei den befreundeten Funktionen sieht man sehr gut, wie die Ableitungen der transzendenten zustandekommen, und wie sie realisiert wurden.
Weiterhin ist bemerkenswert, wie die abgeleitenen transzendenten Funktionen wie beispielsweise $\sinh(x)$ oder $\cot(x)$ von den bereits überladenen Funktionen abgeleitet werden können, und somit mit wenig Mehraufwand sehr viel zusätzliche Funktionalität erreicht wird.
Die von uns auch überladene `pow(x,y)` Funktion sollte mit Vorsicht verwendet werden, da einerseits die Ableitungen meist falsch sind und sie sich in Grenzfällen auch nicht korrekt verhält. Um Potenzen zu behandeln sollte auf $\exp(x)$ und $\ln(x)$ zurückgegriffen werden.

```
/*###########
autodiff.cpp
###########*/

#include "autodiff.h"
#include <sstream>
using namespace std;

/**######################
Schnittstellenfunktionen
######################**/
double autodiff::get_u(){
    return u;
}
double autodiff::get_v(){
    return v;
}
double autodiff::get_w(){
    return w;
}
void autodiff::set_u(double x){
    u = x;
}
void autodiff::set_v(double x){
    v = x;
}
void autodiff::set_w(double x){
    w = x;
}

/**##############################################
Umwandlung des Tripels in einen sinnvollen String
```

```
##############################################**/
string autodiff::toStdString(){
    stringstream str;
    str << "(" << u << "/" << v << "/" << w << ")";
    return str.str();
}


/**########################################
(Kopier-)Kon(De)struktor und Zuweisungsoperator
########################################**/
autodiff::autodiff(double x=0, double y=0, double z=0){
    u = x;
    v = y;
    w = z;
}
autodiff::autodiff(const autodiff &a){
    u = a.u;
    v = a.v;
    w = a.w;
}
autodiff &autodiff::operator =(const autodiff &a){
    u = a.u;
    v = a.v;
    w = a.w;
    return *this;
}

autodiff::~autodiff(){

}

/**##########################################################
Operator * wird mit einer skalaren Multiplikation überladen
##########################################################**/
autodiff operator *(double k, autodiff a){
    autodiff res(k*a.u, k*a.v, k*a.w);
    return res;
}

/**#####################################
Die Potenzierungsfunktion wird überladen
#####################################**/
autodiff pow(autodiff f, autodiff g){
```

```
autodiff res(0,0,0);
if (f.u == 0) {
res.u = 0;

}
res.u = pow(f.u,g.u);
res.v = exp(g.u * log(fabs(f.u)))*(g.v * log(fabs(f.u)) + g.u*(f.v/f.u));
res.w = exp(g.u * log(fabs(f.u))) * (pow((g.v * log(fabs(f.u)) +
            g.u*(f.v/f.u)),2) + g.w*log(fabs(f.u)) + 2*g.v*(f.v/f.u) +
            g.u*((f.w*f.u-f.v*f.v)/(f.u*f.u)));
return res;

}

/**############################################
Die transzendenten Funktionen werden überladen

############################################**/
autodiff exp(autodiff x){ //Exponential-Funktion
    autodiff res(exp(x.u),  exp(x.u) * x.v,  exp(x.u)  * (x.v * x.v + x.w));
    return res;
}
autodiff sin(autodiff x){ //Sinus-Funktion
    autodiff res(sin(x.u),  cos(x.u) * x.v,  x.w * cos(x.u) - sin(x.u) * x.v * x.v);
    return res;
}
autodiff cos(autodiff x){ //Cosinus-Funktion
    autodiff res(cos(x.u),  -sin(x.u)*x.v,  -(cos(x.u)*x.v*x.v + x.w*sin(x.u)));
    return res;
}
autodiff ln(autodiff x){ //Natürlicher Logarithmus
    autodiff res(log(x.u), x.v/x.u, (x.w*x.u - x.v*x.v)/(x.u*x.u));
    return res;
}
autodiff tan(autodiff x){ //Tangens definiert mit den bereits
                          //überladenen Funktionen Sinus und Cosinus
    autodiff res = sin(x)/cos(x);
    return res;
}
autodiff cot(autodiff x){ //Kotangens definiert mit den bereits
                          //überladenen Funktionen Sinus und Cosinus
autodiff res = cos(x)/sin(x);
return res;
}
```

```
/**#####################################################################
Hyperbolische Winkelfunktionen als Komposition der Exponentialfunktion
#####################################################################**/
autodiff cosh(autodiff x){
    autodiff res = 0.5*(exp(x) + exp(-x));
    return res;
}
autodiff sinh(autodiff x){
    autodiff res = 0.5*(exp(x) - exp(-x));
    return res;
}
autodiff tanh(autodiff x){
    autodiff res = sinh(x)/cosh(x);
    return res;
}
autodiff coth(autodiff x){
    autodiff res = cosh(x)/sinh(x);
    return res;
}

/**##################################
Standart-Operatoren werden überladen
##################################**/
autodiff operator + (autodiff a, autodiff b){
    //Bei Addition können die Werte einfach addiert werden.
    autodiff res(a.u + b.u,  a.v + b.v,  a.w + b.w);
    return res;
}
autodiff operator - (autodiff a, autodiff b){
    //Verwendung des bereits überladenen unären Operators (-)
    autodiff res = a + (-b);
    return res;
}
autodiff operator * (autodiff a, autodiff b){
    //Zweifache Anwendung der Produktregel.
    autodiff res(a.u * b.u,  a.u * b.v + a.v * b.u,
                 a.u * b.w + 2 * a.v * b.v + a.w*b.u);
    return res;
}
autodiff operator / (autodiff a, autodiff b){
    //Zweifache Anwendung der Quotientenregel mit Prüfung
    //auf Nullstellen im Nenner.
    if (fabs(b.u) < 0.00001){
```

```
        autodiff res(0,0,0);
        return res;
    }
    else{
        autodiff res(a.u / b.u,  (a.v*b.u-a.u*b.v)/(b.u*b.u),
                    ((a.w*b.u-b.w*a.u)*b.u-
                    ((a.v*b.u)-(b.v*a.u))*2*b.v)/(b.u*b.u*b.u));
     return res;
    }
}

/**************
Unärer Operator
*************/
autodiff operator - (autodiff a){
    autodiff res(-a.u, -a.v, -a.w);
    return res;
}
```

5.3 Anwendung

Um den vom Parser erstellten binären Baum auszuwerten, haben wir eine rekursive Funktion geschrieben, welche den Baum durchläuft und dabei immer die an einem Knoten stehenden Operationen auf die beiden Äste des Knotens anwendet. Hierbei ist die Abbruchbedingung, dass wir bei einem Blatt angelangt sind, was man daran erkennt, dass ein Knoten keine Äste mehr besitzt.
In der switch-Verzweigung sieht man sehr gut, wie die überladenen Operatoren rekursiv angewendet werden. Durch den zweifachen Aufruf von `parser::derive` in jedem Rekursionsschritt liegt hier eine nicht-lineare Rekursion vor.

```
autodiff parser::derive(double x, Baum term){
    if ((term.l == 0)&&(term.r == 0)){
        //Ableitung von der Variable x wird eingesetzt
        if(term.text == "x"){
            autodiff res = X;
            res.set_u(x);
            return res;
        }
            //Konstante e wird korrekt erkannt
            if(term.text == "e"){
            return exp(1)*C;
        }
        //keiner der oberen Fälle ist eingetreten => Konstante
        //multipliziert mit einem Vielfachen wird zurückgegeben
        return atof(term.text.c_str())*C;
    }
    else{
        switch(term.op){
            //Je nach Operator werden die benötigten für autodiff überladenen
            //Freundfunktionen angewendet.
            case 2:
                return pow(derive(x, *term.l),derive(x,*term.r));
            case 3:
                return derive(x,*term.l)*derive(x,*term.r);
            case 4:
                return derive(x,*term.l)/derive(x,*term.r);
            case 5:
                return derive(x,*term.l)-derive(x,*term.r);
            case 6:
                return derive(x,*term.l)+derive(x,*term.r);
            case 20:
                return exp(derive(x,*term.r));
            case 21:
                return sin(derive(x,*term.r));
```

```
            case 22:
                return cos(derive(x,*term.r));
            case 23:
                return ln(derive(x,*term.r));
            case 24:
                return tan(derive(x,*term.r));
            case 25:
                return sinh(derive(x,*term.r));
            case 26:
                return cosh(derive(x,*term.r));
            case 27:
                return tanh(derive(x,*term.r));
            case 28:
            return cot(derive(x,*term.r));
                case 29:
            return coth(derive(x,*term.r));
        }
    }
}
```

6 Beispiele

6.1 Funktions- und Ableitungswerte

Zur Demonstration der grundlegenden Funktionalität unseres Programms wählen wir die Funktion $f(x) = e^{\sin(x)}$. Dazu ist folgende Eingabe notwendig:

```
f(x) = exp(sin(x))
```

Dies liefert uns folgende Ausgabe:

```
> Input interpretation: f(x) = exp(sin(x))

> Funktionswerte und Ableitungen:

     f(3) = 1.15156
    f'(3) = -1.14004
   f''(3) = 0.966121

> Funktionswerte und Ableitungen:

     f(1.5708) = 2.71828
    f'(1.5708) = -9.98481e-06
   f''(1.5708) = -2.71828

> Funktionswerte und Ableitungen:

     f(0) = 1
    f'(0) = 1
   f''(0) = 1
```

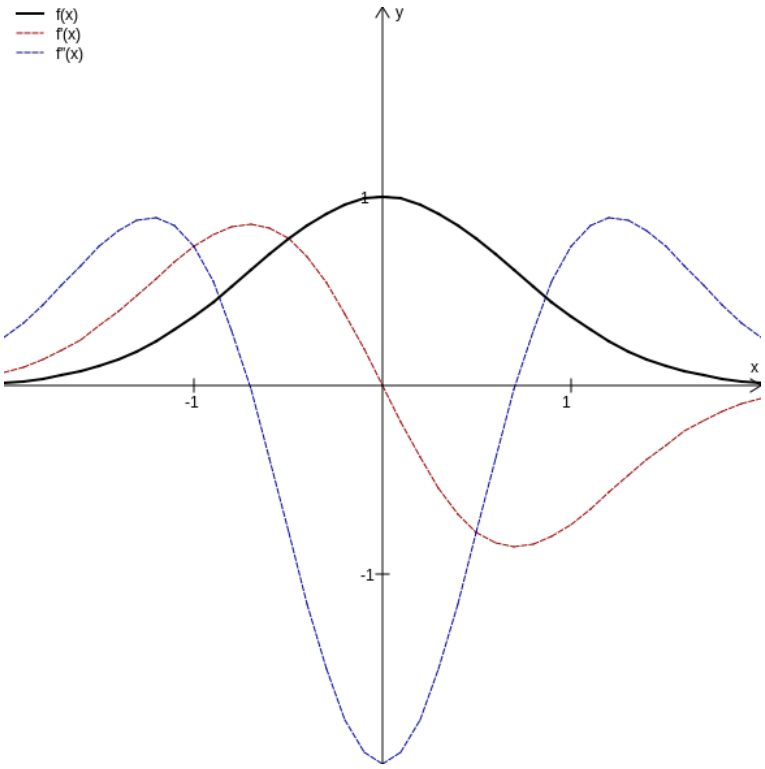
An der Stelle $x \approx \pi/2$ ist die erste Ableitung sehr klein, da $\sin(x)$ an dieser Stelle ein lokales Maximum hat. Und deshalb auch $f(x)$, weil die

6.2 Plot

Um die Funktion des Plottens vorzustellen benutzten wir folgende Eingabe:

```
f(x) = exp(-x*x)
```

Unser Funktionsterm ist also $f(x) = e^{-x^2}$:



Die Funktion selbst stellt die Gauss'sche Glockenkurve dar. Die beiden farbigen Kurven stellen die beiden von `autodiff` berechneten Ableitungen dar.

6.3 Wertetabelle

Um die Funktionalität der Wertetabelle zu veranschaulichen, wählen wir ein Polynom 5-ten Grades:

```
f(x) = x*(x-1)*(x+1)*(x-1)*(x+1)*0.5
```

Unser Funktionsterm ist also $f(x) = \frac{1}{2}x(x+1)^2(x-1)^2$ und die dazu gehörende Ausgabe:

```
> Wertetabelle fuer x = -2..2(0.5):
```

x =	-2	-1.5	-1	-0.5	0	0.5	1	1.5	2
f(x) =	-9	-1.17188	0	-0.140625	0	0.140625	0	1.17188	9
f'(x) =	28.5	6.40625	0	-0.09375	0.5	-0.09375	0	6.40625	28.5
f''(x) =	-68	-24.75	-4	1.75	0	-1.75	4	24.75	68

An der Wertetabelle sieht man sehr gut die doppelten Nullstellen bei -1 und $+1$, da an diesen Stellen sowohl der Funktionswert selbst, als auch die Ableitung 0 sind.

6.4 Newton-Iteration

Für die Newton-Iteration wählen wir eine recht einfache Funktion, die bei $x = 1$ eine Nullstelle haben sollte: $f(x) = e^x - e$. Hierbei kann man zwischen verschiedenen Startwerten und Genauigkeiten wählen:

```
> Newton mit s=4 und Toleranz e=0.1 nach 6 Iterationen:

     x_0 = 1.01099


> Newton mit s=4 und Toleranz e=0.0001 nach 7 Iterationen:

     x_0 = 1.00001
```

Man erkennt, dass eine höhere Forderung an die Genauigkeit erstens mehr Iterationen nach sich zieht, aber logischerweise auch die Nullstelle genauer bestimmt.

7 Literatur

- Helmut Erlenkötter - C++ Objektorientiertes Programmieren von Anfang an
- Arnold Willemer - Einstieg in C++
- Jürgen Wolf - Qt 4.6 GUI-Entwicklung in C++
- Skript Wissenschaftliches Rechnen - Werner Vogt