NEWTONVERFAHREN

Nichtlineare Optimierung ohne Nebenbedingungen

Petkova Neli,Fallosch Nico,Kasic Ajla,Mittendorfer Patrick,Misurec Patrik

October 2015

Contents

1 Introduction **2**

2 Theorie **2**
2.1 Herleitung 2
2.1.1 Nullsetzen der Ableitung 2
2.1.2 Approximation durch quadratisches Polynom 4
2.2 Anwendungsbeispiel 5
2.3 Problemfall 5
2.4 Gedämpftes Newtonverfahren 6
2.5 Mehrdimensionales Newtonverfahren 6

3 Implementation **7**
3.1 Tool für Problemfall 7

4 Homepage **9**

1 Introduction

Das Newtonverfahren (auch Newton-Raphson gennant) ist ein Algorithmus zur Suche von Nullstellen. Es handelt sich dabei um ein Iterationsverfahren, welches sich pro Iteration der Nullstelle immer weiter annähert. Idealerweise wird ein Startpunkt bestimmt, der bereits möglichst nahe an der Nullstelle liegt. Die Funktion wird durch Ihre Tagente in diesem Punkt ersetzt, wobei der Schnittpunkt zwischen Tangente und x-Achse als neuer Ausgangspunkt für das Iterationsverfahren verwendet wird. Dieser Schritt wird analog solange durchgeführt, bis die gewünschte Rechengenauigkeit erreicht ist.

Durch Abwandlung der Formel $x_{k+1} = x_k - \frac{\mathrm{f}(x_k)}{\mathrm{f}'(x_k)}$ welche zur Bestimmung des Nullpunkts einer Funktion verwendet wird, lässt sich auch Minimumx bzw. Maximum der selben Funktion ermitteln. Da man hierbei eigentlich die Nullstelle der Ableitung sucht, ergibt sich für das Verfahren folgende Formel: $x_{k+1} = x_k - \frac{\mathrm{f}'(x_k)}{\mathrm{f}''(x_k)}$

Analog kann der Alogirthmus auch für mehrdimensionale Probleme angewandt werden. Die Formel hierfür lautet: $x_{k+1} = x_k - (H_f(x_k))^{-1}(\nabla F(x_k))'$

$x_{k+1} = x_k - \frac{\mathrm{f}(x_k)}{\mathrm{f}'(x_k)}$

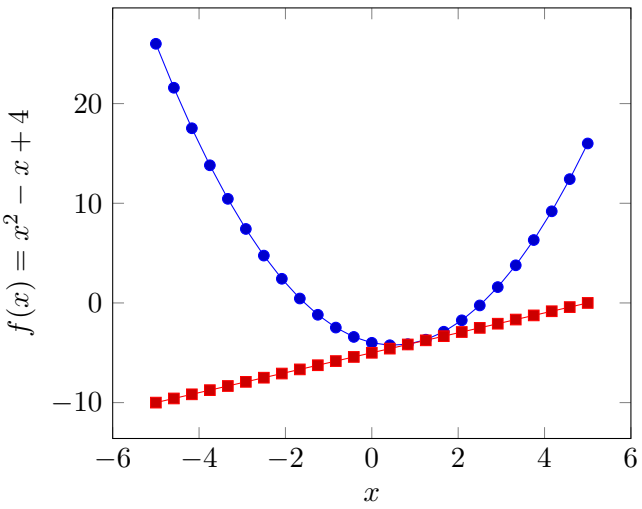
$x_{k+1} = x_k - \frac{\mathrm{f}'(x_k)}{\mathrm{f}''(x_k)}$

$\frac{\mathrm{f}'(x_k)}{\mathrm{f}''(x_k)}\mathrm{x}_{k+1} = x_k - (H_f(x_k))^{-1}(\nabla F(x_k))'$

2 Theorie

2.1 Herleitung

2.1.1 Nullsetzen der Ableitung



Gegeben ist eine Funktion f(x). Angenommen $g(x) = f'(x)$.
Wir wählen einen beliebigen Startpunkt x_0 von g(x).
Im nächsten Schritt sucht man die Steigung der Tagente m = $g'(x_0)$.

m = $\frac{\Delta g(x)}{\Delta x} = \frac{\mathrm{g}(x_0)}{(x_0 - x_1)}$

$g'(x_0) = \frac{\Delta g(x)}{\Delta x} = \frac{\mathrm{g}(x_0)}{(x_0 - x_1)}$

Danach multipliziert man beide Seiten der Gleichung mit $(x_0 - x_1)$.

$g'(x_0) \times (x_0 - x_1) = g(x_0)$

$x_0 - x_1 = \frac{\mathrm{g}(x_0)}{\mathrm{g}'(x_0)}$

Daher kommt raus:

$-x_1 = -x_0 + \frac{\mathrm{g}(x_0)}{\mathrm{g}'(x_0)}$

Man multipliziert die Gleichung mit (-1). Das ergibt:

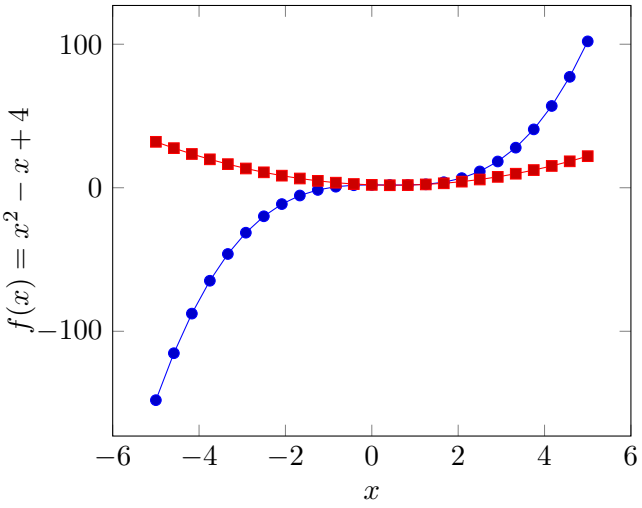
$x_1 = x_0 - \frac{\mathrm{g}(x_0)}{\mathrm{g}'(x_0)}$

Analog, lässt sich die Iterationsformel auf :

$x_{k+1} = x_k - \frac{\mathrm{g}(x_k)}{\mathrm{g}'(x_k)}$ herleiten. Bzw. lösen wir $g(x) = f'(x)$ auf.

$x_{k+1} = x_k - \frac{\mathrm{f}'(x_k)}{\mathrm{f}''(x_k)}$

2.1.2 Approximation durch quadratisches Polynom



Gegeben ist eine Funktion f(x).Das approximierende Polynom sei das Taylorpolynom 2.Grades. Die Taylorformel für ein Polynom 2.Grades lautet allgemein:

$f(x) = f(x_0) + \frac{f'(x_0)}{1!} \times (x_1 - x_0) + \frac{f''(x_0)}{2!} \times (x_1 - x_0)^2$

Das lässt sich allgemein wie folgt darstellen:

$f(x) = a \times x^2 + bx + c$

Erste Ableitung von $f'(x)$:

$f'(x) = 2a \times x + b$

Zweite Ableitung von $f''(x)$:

$f''(x) = 2a$

Die Tiefstelle von $f(x)$ erhält man:

$f'(x*) = 2a \times x + b = 0 => x* = \frac{-b}{(2a)}$

Hier muss darauf geachtet werden, dass a nicht negativ sein darf, weil es sonst das Maximum oder der Wendepunkt wäre.

$f'(x) = 2a + b$

$f''(x) = 2a$

Daher ergibt sich für b und a:

$2a = f''(x)$

$b = -2a \times x + f'(x) = -f''(x) \times x + f'(x)$

Nun setzen wir a und b in x* ein

$x* = \frac{-b}{(2a)} =$

$x* = \frac{-(-f''(x) \times x + f'(x))}{2 \times \frac{f''(x)}{2}} = \frac{f''(x) \times x - f'(x)}{f''(x)} = x - \frac{f'(x)}{f''(x)}$

Das kann man nun rekursiv fortsetzen:

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

2.2 Anwendungsbeispiel

2.3 Problemfall

Gegeben sei die Funktion $f(x) = \frac{x^4}{4} - x^2 + 2x$ und der Startpunkt $x_0 = 0$. Bei Anwendung des Newtonverfahrens zur Bestimmung des Minimus der Funktion ergeben sich folgende Iterationsschritte:

$f(x) = \frac{x^4}{4} - x^2 + 2x$
$f'(x) = x^3 - 2x + 2$
$f''(x) = 3x^2 - 2$

$x_0 = 0$

$x_1 = 0 - \frac{0^3 - 2 \times 0 + 2}{3 \times 0^2 - 2} = 1$

$x_2 = 1 - \frac{1^3 - 2 \times 1 + 2}{3 \times 1^2 - 2} = 0$

$x_3 = 0 - \frac{0^3 - 2 \times 0 + 2}{3 \times 0^2 - 2} = 1$

$x_4 = 1 - \frac{1^3 - 2 \times 1 + 2}{3 \times 1^2 - 2} = 0$

Wie man hier feststellen kann, alternieren die Werte zwischen 0 und 1, wodurch das Verfahren nicht zum Minimum der Funktion konvergiert.

2.4 Gedämpftes Newtonverfahren

Beim gedämpften Newtonverfahren wird die Iterationsformel um einen Dämpfungsparameter λ erweitert, um so die Weite des Intervalles verändern zu können.

$$x_{k+1} = x_k - \lambda \times \frac{\mathrm{f}'(x_k)}{\mathrm{f}''(x_k)}$$

Beim Versuch die Funktion aus dem vorherigen Kapitel mit Hilfe des gedämpften Newtonverfahren unter Zuhilfename eines selbst geschrieben C++ Programmes zu lösen, ergaben sich folgende Ergebnisse:

$\lambda = 0.5, x_0 = 0 \rightarrow 64$ Schritte

$\lambda = 0.25, x_0 = 0 \rightarrow 245$ Schritte

$\lambda = 0.25, x_0 = 0 \rightarrow 47$ Schritte

$\lambda = 0.95, x_0 = 0 \rightarrow 39$ Schritte

$\lambda = 0.99, x_0 = 0 \rightarrow \infty$ Schritte

Daraus lässt sich schließen, dass die Gleichung unter Verwendung eines geeigneten λ und der Bedingung $x_0 = 0$ mittles gedämpften Newtonverfahrens lösbar ist. Beim Versuch λ nahe an eins zu wählen, konnten wir keine Ergebnisse erhalten.

2.5 Mehrdimensionales Newtonverfahren

Der lokale Konvergenzbeweis kann auf die gleiche Weise auch im mehrdimensionalen Newtonverfahren geführt werden, zwar durch Anwenden von Jakobi Matrix.

Analog zu eindimensionalem Newtonverfahren, erhaltet man die Formel für mehrdimensionalen Funktionen:

$x_{k+1} = x_k - \frac{\bigtriangledown F(x_k)}{(H_f(f_k))}$

Nun führen wir die Berechnung für das lokale Extremum von f(x, y) auf das Gleichungssystem:

$\frac{\sigma}{\sigma x} f(x, y) = 0$

$\frac{\sigma}{\sigma y} f(x, y) = 0$

daraus folgt ein Gleichungssystem, das mittels Jakobi Matrix zu lösen ist:

$$f1(x, y) = 0$$

$$f2(x, y) = 0$$

Dies kann iterativ erfolgen. In jedem Schritt ist ein lineares Gleichungssystem zu lösen

$$\begin{bmatrix} \frac{\sigma f1(x_k,y_k)}{\sigma x} & \frac{\sigma f1(x_k,y_k)}{\sigma y} \\ \frac{\sigma f2(x_k,y_k)}{\sigma x} & \frac{\sigma f2(x_k,y_k)}{\sigma y} \end{bmatrix} \begin{bmatrix} z1 \\ z2 \end{bmatrix} = \begin{bmatrix} -f1(x_k, y_k) \\ -f2(x_k, y_k) \end{bmatrix} \text{ Startvektor ist : } \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

Das lokale Extremum von f(x, y, z) findet man mittels Jakobi Matrix

$$\frac{\sigma}{\sigma x} f(x, y, z) = 0$$

$$\frac{\sigma}{\sigma y} f(x, y, z) = 0$$

$$\frac{\sigma}{\sigma z} f(x, y, z) = 0$$

woher ein Gleichungssystem folgt:

f1(x, y, z) = 0
f2(x, y, z) = 0
f3(x, y, z) = 0

$$\begin{bmatrix} \frac{\sigma f1(x_k,y_k)}{\sigma x} & \frac{\sigma f1(x_k,y_k)}{\sigma y} & \frac{\sigma f1(x_k,y_k)}{\sigma z} \\ \frac{\sigma f2(x_k,y_k)}{\sigma x} & \frac{\sigma f2(x_k,y_k)}{\sigma y} & \frac{\sigma f2(x_k,y_k)}{\sigma z} \\ \frac{\sigma f2(x_k,y_k)}{\sigma x} & \frac{\sigma f2(x_k,y_k)}{\sigma z} & \frac{\sigma f3(x_k,y_k)}{\sigma z} \end{bmatrix} \begin{bmatrix} z1 \\ z2 \\ z3 \end{bmatrix} = \begin{bmatrix} -f1(x_k, y_k) \\ -f2(x_k, y_k) \\ -f3(x_k, y_k) \end{bmatrix}$$

3 Implementation

3.1 Tool für Problemfall

Um die in Kapitel 2.3 beschriebene "Problemgleichung" $f(x) = \frac{x^4}{4} - x^2 + 2x$ mit Startpunkt $x_0 = 0$ besser analysieren zu können, haben wir ein Testtool in C++ geschrieben. Das Tool wendet das gedämpfte Newtonverfahren an. Der Benutzer hat weiters die Möglichkeit, den Startpunkt, λ und die maximale Anzahl an Iterationsschritten zu wählen. Bei Verwendung von $\lambda = 1$ entspricht das Verfahren dem regulären Newtonverfahren.

```
1 #include "stdafx.h"
```

```
2  #include <math.h>
3  #include <iostream>
4
5  double point = 0;
6  double lambda = 0.5;
7  int maxRuns = 500;
8
9  int main()
10 {
11    //Calculator: f(x) = x^4/4-x+2x
12    //öLsung mit ägedmpften Newtonverfahren
13
14    //Info:
15    //Wenn Lambda in starker äNhe von 1 ist, tritt ein endloser Zyklus ein↩
          vermeidet die Konvergenz
16
17
18    char c;
19    std::cout << "y for default values / n to input values" << std::endl;
20    std::cin >> c;
21
22    if (c == 'n')
23    {
24       std::cout << "enter startpoint (double): ";
25       std::cin >> point;
26
27       std::cout << std::endl << "enter lambda (double): ";
28       std::cin >> lambda;
29
30       std::cout << std::endl << "enter maxRuns int: ";
31       std::cin >> maxRuns;
32
33       std::cout << std::endl << std::endl;
34
35    }
36    for (int i = 0; i < maxRuns; i++)
37    {
38       if (point == point - lambda * ((pow(point, 3) - 2 * point + 2) / (3↩
             * (pow(point, 2)) - 2)))
39       {
40          std::cout << "Done after: " << i+1 << " runs" << std::endl;
41          break;
42       }
43       point = point - lambda * ((pow(point, 3) - 2 * point + 2) / (3 * (↩
             pow(point, 2)) - 2));
44       std::cout << point << std::endl;
45    }
46    std::cin >> point;
47     return 0;
48 }
```

4 Homepage

Unsere HomePage ist unter http://ops-newton.github.io/Page aufrufbar.