Rheinisch-Westfälische Technische Hochschule
Institut für Geometrie und Praktische Mathematik

Mathematisches Praktikum (MaPra) — Sommersemester 2011

Prof. Dr. Wolfgang Dahmen — M.Sc. Mathieu Bachmann, Dipl.-Math. Jens Berger, M.Sc. Liang Zhang

Aufgabe 1 (für Informatiker)

Bearbeitungszeit: eineinhalb Wochen (Abgabe bis Montag, den 2. Mai 2011)

Testattermin: Donnerstag, der 5. Mai 2011

Mathematischer Hintergrund: Nullstellenbestimmung mit einschlusserhaltenden Verfahren

Elemente von C++: Schleifen, case–Anweisung, Funktionen, Ein-/Ausgabe, Einbinden einer Grafikbibliothek

12 Punkte

Aufgabenstellung

Gegeben seien $x_0, x_1 \in \mathbb{R}$ und eine stetige Funktion $f : [x_0, x_1] \to \mathbb{R}$ mit genau einer Nullstelle, so dass $f(x_0)$ und $f(x_1)$ verschiedene Vorzeichen haben. Schreiben Sie ein Programm, das die Nullstelle mit Hilfe der vier im folgenden Text beschriebenen Methoden berechnet.

Iterative Verfahren, Ordnung und Effizienzindex

Die nachfolgenden Verfahren sind sogenannte *iterative* Verfahren, die aus einer oder mehreren Näherungslösungen jeweils eine neue — hoffentlich bessere — Näherung berechnen. Mehrfache Anwendung (Iteration) dieser Verfahren liefert dann eine beliebig gute Näherung. Die *Ordnung* p eines Verfahrens gibt an, wie sich der Fehler $\mathrm{err}_i := |x_i - \bar{x}|$ zwischen der exakten Lösung $\bar{x}$ und der Näherungslösung $\bar{x}_i$ *pro Verfahrensschritt* entwickelt: $\mathrm{err}_{i+1} \leq c \cdot \mathrm{err}_i^p$ mit einer Konstante $c > 0$. Der *Effizienzindex* (nach Traub [3]) berücksichtigt darüber hinaus, wie groß der Aufwand für einen solchen Schritt ist, indem angegeben wird, wie sich der Fehler *pro Funktionsauswertung* entwickelt. Der Effizienzindex ist daher $q = p^{1/k}$, wobei k die Anzahl der Funktionsauswertungen pro Verfahrensschritt ist. In dem zu schreibenden Programm werden die Geschwindigkeitsunterschiede von Verfahren mit unterschiedlichem Index deutlich.

Einschlusserhaltende Verfahren

Ausgehend von den zwei Startwerten x_0 und x_1 und den zugehörigen Funktionswerten $f_i := f(x_i)$ sollen x_i $(i = 2, 3, \ldots)$ so bestimmt werden, dass die jeweils letzten beiden Iterierten die Nullstelle einschließen, dass also die zugehörigen Funktionswerte unterschiedliche Vorzeichen haben. Dazu wird zu je zwei Iterierten x_{i-1} und x_i die Gerade durch die Punkte (x_{i-1}, f_{i-1}) und (x_i, f_i) gelegt. Der Schnittpunkt dieser Geraden mit der x–Achse sei x_{i+1}. Er liegt zwischen x_{i-1} und x_i, weil f_{i-1} und f_i unterschiedliche Vorzeichen haben. Ist $f_{i+1} := f(x_{i+1}) = 0$, so hat man die Nullstelle gefunden und kann die Iteration abbrechen. Dieses Verfahren ist das *Sekantenverfahren*, welches aber nicht unbedingt die Nullstelle einschließt, da nicht garantiert ist, dass f_i und f_{i+1} verschiedene Vorzeichen haben. Daher modifizieren wir es wie folgt: Liefern x_i und x_{i+1} keinen Einschluss, so überschreiben wir die Werte x_i und f_i durch x_{i-1} und $\gamma_i f_{i-1}$ mit dem Parameter $\gamma_i > 0$. So wird erzwungen, dass x_i und x_{i+1} einen Einschluss liefern. Der Parameter γ_i beeinflusst, wo der Schnittpunkt der Geraden mit der x–Achse im nächsten Schritt liegt, und hat erheblichen Einfluss auf den Effizienzindex. Je nach Wahl von γ_i unterscheiden wir die folgenden vier Verfahren:

- **Regula Falsi**, $\gamma_i = 1$ (teilweise auch Primitivform der Regula Falsi genannt):
  Dies ist das einfachste der hier verwendeten Verfahren, hat aber den Nachteil, dass man (vgl. Skizze) fast immer auf einer Seite mit einem x–Wert „hängen bleibt“ und sich nur der andere x–Wert der Lösung annähert. Dies ist für den relativ schlechten Effizienzindex $q = 1$ verantwortlich.

Das „Hängenbleiben“ bei einem x–Wert resultiert aus der Tatsache, dass der Schnitt mit der Geraden in allen weiteren Schritten nie zwischen der Nullstelle und diesem x–Wert liegt. Dies soll nun durch Verkleinerung des Betrags des zugehörigen Funktionswerts $f(x)$ erreicht werden, weil so der Schnittpunkt der Geraden mit der x–Achse in Richtung von x verlagert wird.

- **Illinois–Verfahren**, $\gamma_i = \frac{1}{2}$:
  Dieser einfache Trick erhöht den Effizienzindex des Verfahrens auf $q = \sqrt[3]{3} \approx 1.442$, was eine erhebliche Verbesserung darstellt.

Eine weitere Verbesserung lässt sich erreichen, indem man γ_i in Abhängigkeit von den bisher errechneten Werten wählt:

- **Pegasus–Verfahren** [2], $\gamma_i = \frac{f_i}{f_i + f_{i+1}}$:
  Der Parameter γ_i ist dabei so gewählt, dass $\frac{\gamma_i}{1-\gamma_i} = \frac{f_i}{f_{i+1}}$ gilt. Hier ist der Effizienzindex bereits auf $q \approx 1.642$ gestiegen. Damit haben wir die Effizienz des Sekantenverfahrens ($q = \frac{1+\sqrt{5}}{2} \approx 1.618$ — der goldene Schnitt lässt grüßen) bereits übertroffen, ohne den Einschluss (im Gegensatz zum Sekantenverfahren) verlieren zu können.

- **Anderson–Björck–Verfahren** [1]:
  Hier liegt folgende Idee zugrunde: Liefern x_i und x_{i+1} keinen Einschluss, so wird vom Punkt x_{i+1} ein Newton–Schritt mit der durch (x_{i-1}, f_{i-1}), (x_i, f_i) und (x_{i+1}, f_{i+1}) gegebenen Parabel durchgeführt, um x_{i+2} zu erhalten. Das Verblüffende ist nun, dass sich dieser Schritt in unsere obige Terminologie einpassen lässt. Wertet man nämlich die beim Newton–Schritt verwendete Tangente an der Stelle x_{i-1} aus, so erhält man das Ergebnis
  $$\left(1 - \frac{f_{i+1}}{f_i}\right) f_{i-1}.$$
  Wählt man also $\gamma_i = 1 - \frac{f_{i+1}}{f_i}$, so entspricht die Gerade durch die Punkte (x_{i+1}, f_{i+1}) und $(x_{i-1}, \gamma_i\, f_{i-1})$ genau der Tangente. Der Newton–Schritt wird auf diese Weise gewissermaßen automatisch bei der nächsten Iteration durchgeführt. Sollte $\gamma_i \leq 0$ sein, so wird als *Notschritt* ein Illinois–Schritt gemacht, also $\gamma_i = 0.5$ gesetzt. Das so entstehende Verfahren hat den Effizienzindex $q \approx 1.710$ bzw. $q \approx 1.682$, je nach Ablauf der Iteration.

Abbruchbedingung, Funktionsauswertungen

Die Iteration soll beendet werden, wenn $|f(x_{i+1})| \leq$ `eps` und (außer bei der Regula Falsi) $|x_{i+1} - x_i| \leq$ `eps` erfüllt sind. Auch wenn Sie die Nullstelle treffen, also bei $f(x_{i+1}) = 0$, soll die Iteration beendet werden. Die Funktion f sollte an jeder Stelle höchstens einmal aufgerufen werden, um keine unnötige Rechenzeit zu verschwenden. Um außerdem nicht überflüssigen Speicher zu belegen, sollten Sie Ihr Programm so anlegen, dass Sie keine Felder (Arrays) benötigen.

Schnittstellen

Die Startwerte, die Funktion f etc. werden Ihrem Programm durch `unit2.h` und `unit2.o` zur Verfügung gestellt, ebenso Unterprogramme zur grafischen Aufbereitung und Kontrolle der errechneten Werte. Im einzelnen enthält die Header–Datei `unit2.h` folgende Schnittstellen:

```
extern const int AnzahlBeispiele;

extern double eps;
```

```
enum { RegFalsi, Illinois, Pegasus, AndBjrk };

double f ( double x );
```

Die Aufzählung `enum` dient lediglich dazu, den Verfahren Namen zu geben. Das Programm gestaltet sich erheblich übersichtlicher, wenn das Verfahren `Pegasus` statt des Verfahrens `2` aufgerufen wird. Die Konstante `AnzahlBeispiele`, die Variable `eps` und die Funktion `f` erklären sich selbst. Daneben gibt es noch drei Funktionen, die für die Auswahl des Beispiels sowie die Beurteilung und gegebenenfalls die grafische Aufarbeitung der Ergebnisse sorgen:

```
void Start ( int Bsp, double &x0, double &x1, bool Grafik=true, int Pause=300 );
```

Über diese Funktion teilen Sie der Praktikumsumgebung zunächst mit, welches Beispiel `Bsp` Sie rechnen wollen. In den Variablen `x0` und `x1` werden Ihnen dann die Startwerte für die Iteration übergeben. Bei `Grafik=true` erhalten Sie im weiteren Verlauf noch eine grafische Ausgabe der Ergebnisse, mit `Grafik=false` können Sie diese abschalten. Mit dem Parameter `Pause` geben Sie die Zeit in Millisekunden an, die zwischen zwei grafischen Ausgaben gewartet wird. Wenn Sie die Funktion mit nur drei Parametern aufrufen, werden automatisch `Grafik=true` und `Pause=300` gesetzt (Default–Werte). Die Funktion

```
void Ausgabe ( int Verfahren, double xi, double fi, double xiPlus1, double fiPlus1 );
```

muss nach `Start` und nach jeder Iteration aufgerufen werden, um die Zwischenergebnisse zu überprüfen und die Grafik zu aktualisieren. Mit dem Parameter `Verfahren` geben Sie an, welches der vier obigen Verfahren eingesetzt wird (statt der Namen können Sie alternativ auch die Werte `0` bis `3` verwenden). Am Ende des Programms überprüft

```
void Ergebnis ( int Verfahren, double x );
```

die Näherung für die berechnete Nullstelle und schließt, falls nötig, die Grafikausgabe ab.

Hinweise zum Programmablauf

Schreiben Sie Ihr Programm in die Datei `meina2.cpp`. Schreiben Sie das Programm so, dass es den Benutzer zunächst nach der Nummer des zu rechnenden Beispiels fragt. Geben Sie ihm an dieser Stelle die Möglichkeit, das Programm durch Eingabe einer `0` zu beenden. Nachdem Sie `Start` aufgerufen haben, fragen Sie den Benutzer, welches Verfahren er verwenden möchte. Anschließend berechnen Sie dann mit dem angegebenen Verfahren die Nullstelle. Nachdem das Ergebnis überprüft worden ist, kehren Sie zur ersten Eingabe zurück. Versuchen Sie, Ihr Programm möglichst robust gegen fehlerhafte Eingaben zu machen.

Hinweise zum Compilieren und Linken

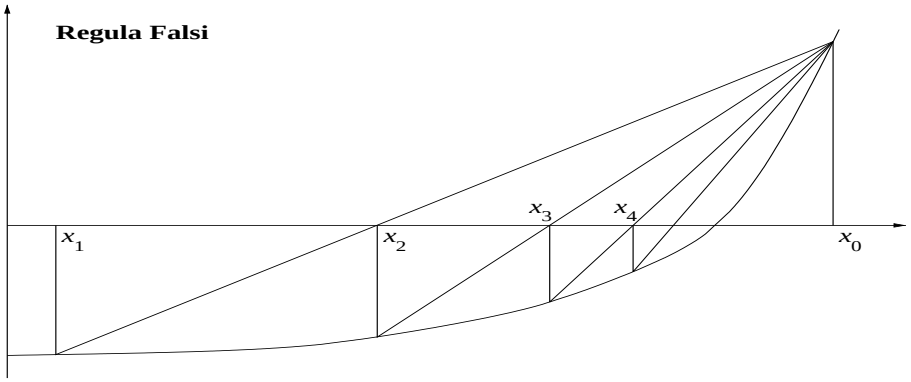
Hier gelten die gleichen Voraussetzungen wie bei Aufgabe 3 aus dem regulären Übungsbetrieb.

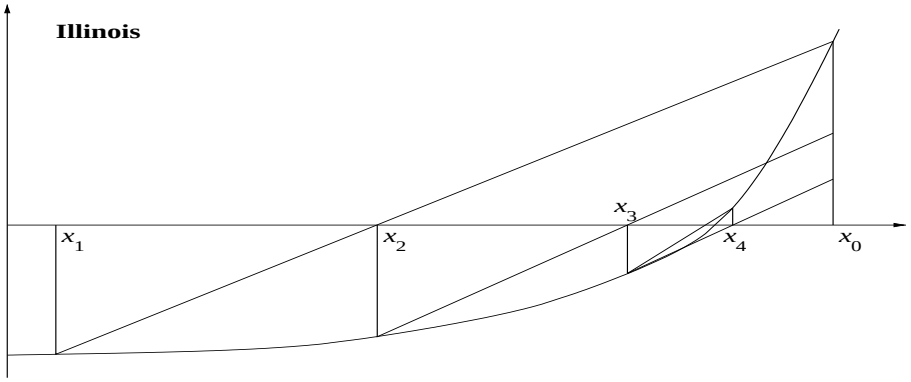
Literatur

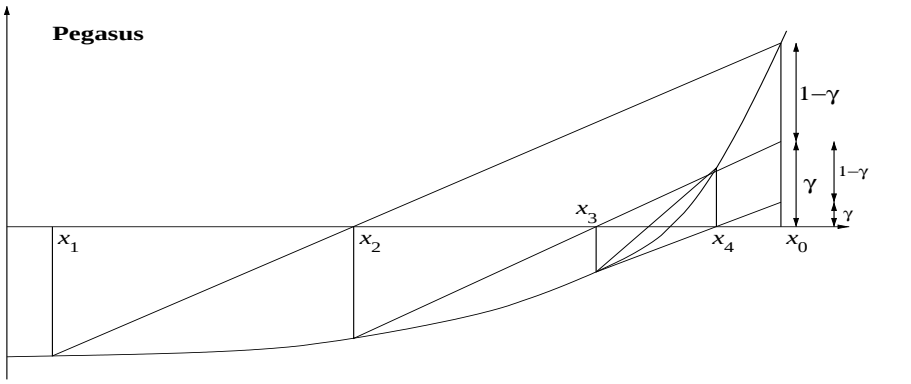
[1] Anderson, N. und Å. Björck: *A new high Order Method of Regula Falsi Type for Computing the Root of an Equation.* BIT, 13:253–264, 1973.

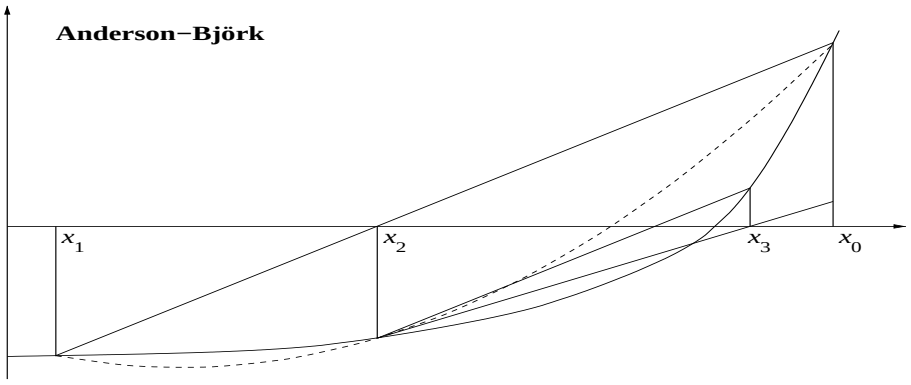
[2] Dowell, M. und P. Jarret: *The "Pegasus" Method for Computing the Root of an Equation.* BIT, 12:503–508, 1972.

[3] Traub, J. F.: *Iterative Methods for the Solution of Equations.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1964.


Regula Falsi
x_1
x_2
x_3
x_4
x_0


Illinois
x_1
x_2
x_3
x_4
x_0


Pegasus
$1-\gamma$
γ
$1-\gamma$
γ
x_1
x_2
x_3
x_4
x_0


Anderson−Björk
x_1
x_2
x_3
x_0