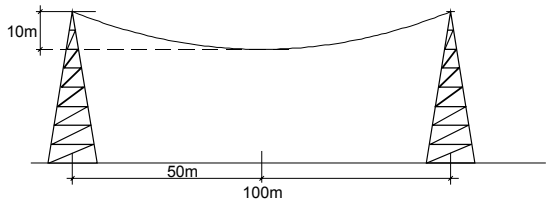
5 Numerische Lösungsverfahren für Gleichungen

Numerische Verfahren können auch Gleichungen und Gleichungssysteme lösen, für die keine analytischen Lösungen bekannt sind. Im Regelfall liefern diese Verfahren aber nur Näherungswerte, die aber beliebig genau berechnet werden können.

Beispiel:
Eine elektrische Leitung wird zwischen zwei gleich hohen Masten, die 100m auseinander liegen, aufgespannt. Der maximale Durchhang der Leitung beträgt in der Mitte 10m. Wie gross ist die Länge *l* der Leitung?



Wir haben hier eine sog. Kettenlinie (cosh *x*).Die Höhe kann durch die Gleichung $y = \lambda \cosh(\frac{x}{\lambda})$ beschrieben werden, wobei λ ein zu bestimmender Parameter ist. Die Randbedingung ist, dass $y(50) = y(0) + 10$ gelten muss. Somit wird die zu lösende Gleichung: $\lambda \cosh\left(\frac{50}{\lambda}\right) = \lambda + 10$. Die Lösung hierfür ist 126.632, wie wir mit den nachfolgend gezeigten Methoden feststellen werden. Mit der Definition der Kettengleichung bestimmen wir dann die Länge *l*=102.619m.

Wie bereits erwähnt, liefern die meisten numerischen Methoden nur Näherungslösungen. Sie können aber (fast) beliebig genau berechnet werden, so dass die Genauigkeit in der Praxis kein Problem darstellt.

5.1 Arbeitsweise der Verfahren

Die meisten numerischen Verfahren arbeiten iterativ. Das heisst, die Lösung wird schrittweise bis zur gewünschten Genauigkeit entwickelt. Praktisch erfolgt dies so, dass im ersten Schritt eine grobe Näherung bestimmt wird und durch geeignete mehrfache Anwendung des Verfahrens die Lösung verbessert wird.

Der Schwerpunkt liegt in diesem Kapitel darin so viel Theorie zu vermitteln, dass die entsprechenden Verfahren in ihrer Wirkungsweise verstanden werden. Nur wenn die Arbeitsweise des Verfahrens verstanden wurde, kann es auch effizient in einer Programmiersprache implementiert werden.

Ein Tip zur praktischen Arbeit, der auch in anderen Bereichen Gültigkeit hat: Ein neues Verfahren zuerst einmal von Hand an einem überschaubaren Kleinbeispiel nachzuvollziehen um die Arbeitsweise zu verstehen.

5.2 Bestimmen von Nullstellen

Das klassische Verfahren zur Lösung von Gleichungen ist die Methode der Bestimmung der Nullstellen. Soll beispielsweise die Gleichung

$$x^7 - \sin x = 18.5$$

gelöst werden, so schreibt man diese um zu

$$f(x) = x^7 - \sin x - 18.5$$

und sucht die Nullstelle(n) dieser Funktion $f(x) = 0$.

Es existieren hunderte von numerischen Verfahren die Lösungen anbieten. Viele sind sehr speziell und nur geeignet um ganz bestimmte Funktionstypen zu bearbeiten.

Die beiden bekanntesten universell einsetzbaren numerischen Methoden sind:

1. ` Methode der Intervallhalbierung
2. ` Nullstellensuche nach Newton

Diese Verfahren bestimmen **beliebig genau** eine reelle Nullstelle für eine gegebene Funktion mit Hilfe eines Iterationsverfahrens. Sie eignen sich praktisch für alle Klassen von Funktionen (Polynome, rationale, trigonometrische, hyperbolische Funktionen, etc.).

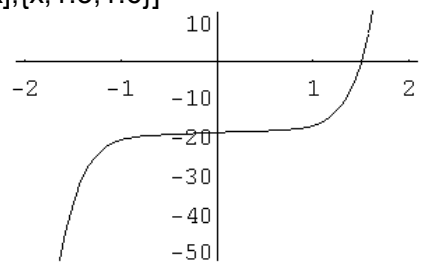
Bei den oben erwähnten numerischen Verfahren erfolgt die Suche von einem Startwert aus mit einem systematischen Iterationsverfahren bis eine Lösung gefunden wird. Ein Problem ist bei allen Verfahren, dass zwar eine Lösung gefunden werden kann, diese aber nicht die einzigste Lösung sein muss. So stellt sich die Frage wie viele Lösungen überhaupt existieren, oder falls keine Lösung gefunden wird, ob das System tatsächlich keine Lösung besitzt.

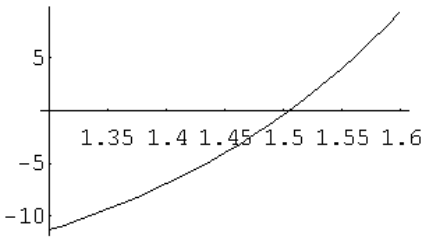
Die Beurteilung der Menge der Lösungen kann sicher aufgrund der Funktionseigenschaften teilweise beantwortet werden. Dies bringt aber in der Praxis eher wenig für eine rasche und einfache Beurteilung.

Der einfachste Weg ist sicher das Aufzeichnen des Graphen der Funktion. Man kann damit aus dem Funktionsverlauf leicht bestimmen wo die reellen Lösungen zu vermuten sind und wie gross die Anzahl ist. Hierzu eignen sich heute vor allem die Mathematikpakete wie Maple, Mathematica, etc.

Beispiel in Mathematica: Lösen der Gleichung $x^7 + \sin x = 18.5$

```
f[x_]:=x^7 +Sin[x]-18.5
Plot[f[x],{x,-2,2}]
Plot[f[x],{x,1.3,1.6}]
```

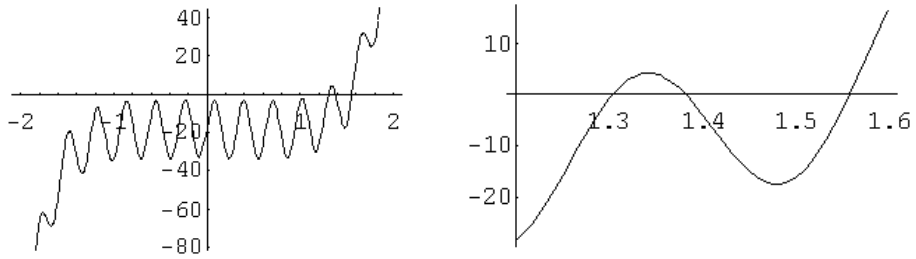




Wir sehen, dass der Graph die x-Achse bei x≈1.51 schneidet und dass dies der einzige Schnittpunkt mit der x-Achse ist. Die grafische Lösung ist hier also 1.51.

Ganz anders sähe die Lösung für $x^7 + 15\sin(20x) = 18.5 \Rightarrow f(x) = x^7 + 15\sin(20x) - 18.5$ aus. Hier wird dann die Lösung nicht mehr eindeutig, sondern wir haben hier drei mögliche Werte für x, die die Gleichung erfüllen:

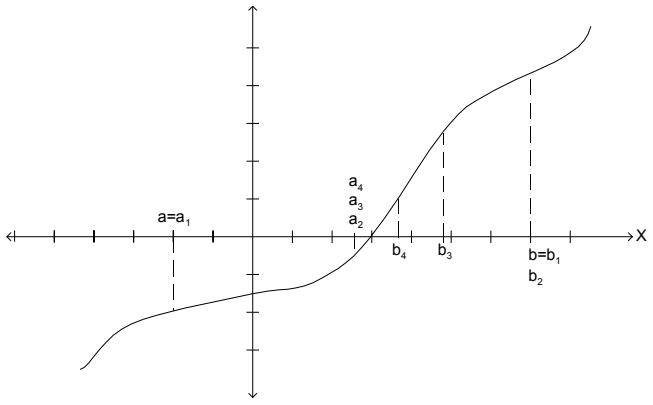
```
f[x_]:=x^7 +15 Sin[20 x]-18.5
Plot[f[x],{x,-2,2}]
Plot[f[x],{x,1.3,1.6}]
```



Wir wollen nachfolgend aber andere Methoden zeigen um die konkreten Zahlenwerte zu bestimmen. Trotzdem bleibt der Graph ein wichtiges Kontrollinstrument zum Verifizieren der Lösungen und zur Vorgabe von Startwerten für die Lösungssuche.

5.2.1 Methode der Intervallhalbierung

Das Verfahren der Intervallhalbierung ist zur Bestimmung der Nullstellen für stetige Funktionen geeignet.
Das Bestimmen einer Nullstelle einer Funktion *f(x)* entspricht der Lösung der Gleichung *f(x)=0*. Ist das Intervall [*a,b*] bekannt indem sich die Nullstelle befindet, so kann dieses halbiert werden. Aufgrund des Funktionswertes an der Halbierungsstelle kann entschieden werden, ob sich die Nullstelle in der oberen oder unteren Hälfte befindet. Dieses neue Intervall wird nach dem gleichen Verfahren wieder halbiert. Dieses auch als *Bisektion* bekannte Verfahren wird sooft wiederholt, bis das Intervall genügend klein ist:



Die Lösung der Gleichung f(*x*)=0 liegt nun in diesem Intervall und kann mit beliebiger Genauigkeit bestimmt werden. Als Startwert muss bei diesem Verfahren allein das Intervall [*a,b*], indem die Nullstelle liegt, bekannt sein.
Praktisch kann mit dieser Methode eine Genauigkeit von ca. 10^{-5} mit einfacher Genauigkeit (float) und ca. 10^{-10} mit doppelter Genauigkeit (double) auf einem PC erreicht werden.

5.2.1.1 Mathematische Grundlagen:

Nach dem **Zwischenwertsatz von R. Bolzano** für stetige Funktionen gilt:
Ist $f:[a,b]\rightarrow B\subset\square$ stetig, so gibt es zu jedem y zwischen $f(a)$ und $f(b)$ mindestens ein $z\in[a,b]$ mit $f(z)=y$.
("f nimmt jeden Zwischenwert zwischen $f(a)$ und $f(b)$ an.")

Daraus folgt insbesondere das Korollar:
Ist $f:[a,b]\rightarrow B\subset\square$ stetig und $f(a)\cdot f(b)<0$ so gibt es mindestens ein $z\in[a,b]$ mit $f(z)=0$.

Da nach Vorgabe im Intervall $[a,b]$ eine Nullstelle liegt, ist sichergestellt, dass wir durch sukzessive Intervallhalbierung ein Intervall $I=[c-\varepsilon/2,c+\varepsilon/2]$ erreichen werden mit $f(c)=0$ und ε beliebig klein.

5.1.1.2 Formulierung im Pseudocode

Pseudocode ist eine grobe, weitgehend Programmiersprachenunabhängige Formulierung des Ablaufes. Pseudocode **ist ein Entwurfshilfsmittel** und dient dazu teilweise komplexe Aufgabenstellung in einer ersten Stufe für die Codierung vorzubereiten. Pseudocode ist gut dazu geeignet in Dokumentationen die wesentlichen Abläufe zu beschreiben. Obwohl Pseudocode sehr verbreitet ist, gibt es keine verbindliche Norm. Das allgemeine Erscheinungsbild ist aber allgemein PASCAL-orientiert mit gewissen Abstrichen:
Man beschränkt sich hier auf die wesentlichen Strukturen:

Sequenz (Aufeinanderfolgende Instruktionen):
Aufeinander folgende, für die Aufgabe wichtige Operationen werden als Instruktionen ausformuliert. Komplexe Operationen können hierbei auch funktionell abstrakt mit eckigen Klammern (Bsp. [Matrix invertieren]) notiert werden. Zuweisungen werden mit dem Pfeil ← getätigt.

Alternative (Bedingte Ausführung aufgrund eines Entscheides):
Je nach Resultat eines Entscheides wird eine Instruktion oder ein ganzer Block ausgeführt.

Der Konstrukt ist

```
IF Bedingung
   THEN
     Instruktion1
   ELSE
     Instruktion 2
END IF
```

Beispiel:

```
IF a==2 THEN
     c ← a^2
  ELSE
     c ← 4
     output a
END IF
```

Iteration (Wiederholtes Ausführen von Instruktionen oder ganzen Programmteilen)
Aufgrund einer Bedingung wird eine Instruktion oder ein Block mehrfach (0..*n*-mal) ausgeführt. Iterationen heissen auch 'Schleifen'. Je nachdem ob die Bedingung (Laufkriterium) am Anfang oder am Ende eines Schleifendurchlaufes geprüft wird, spricht man von vorabprüfenden oder nachprüfenden Schleifen.

Pseudocode benutzt folgenden Konstrukte:

```
FOR Laufvariable=Startwert TO Endwert DO
   Instruktion
END FOR
```

```
WHILE Bedingung DO
   Instruktion
END WHILE
```

```
DO
   Instruktion
WHILE Bedingung
END DOWHILE
```

Beispiel:
```
FOR  i=0 TO 10
  output i*2
END FOR
```

Wesentlich ist beim Pseudocode die *funktionale Beschreibung*. Es werden nur für die Funktion wichtigen Teile konkret formuliert. So sind beispielsweise in Pseudocode Ein- und Ausgaben anonym mit input und output spezifiziert.

5.1.1.3 Arbeitsweise der Methode der Intervallhalbierung

Als Vorgaben müssen zwei Startwerte a und b bekannt sein, die das Suchintervall definieren , wo die Nullstelle zu bestimmen ist.

Daraus folgt die Bedingung, dass mit $u=f(a)$ und $v=f(b)$ gelten muss: $uv < 0$. Das Verfahren bestimmt nun die Mitte des Suchintervalles $c=½ \cdot(a+b)$ und setzt $w=f(c)$.
Danach wird geprüft ob $uw < 0$ oder $vw < 0$ ist. Je nach dem Resultat wird dann das neue, halbierte Suchintervall $[a,c]$ oder $[c,b]$ für einen weiteren Durchlauf genommen. Das Verfahren wird m mal wiederholt. Mit jedem Durchlauf steigt die Genauigkeit der berechneten Näherung für die Nullstelle. Eine Betrachtung zur Genauigkeit des Verfahrens wird im Kapitel "Konvergenzverhalten der Methode der Intervallhalbierung" durchgeführt.

Wir machen nun einen Entwurf des Verfahrens, indem wir die wesentlichen Aktionen der Methode im Pseudocode formulieren. Zusätzlich ist es sicher sinnvoll zuerst zu prüfen, ob die Vorzeichenbedingung für $f(a)f(b)<0$ erfüllt ist. Ist dies nämlich nicht der Fall, findet das Verfahren keine Lösung und läuft einfach m mal durch ohne eine Lösung zu finden.

```
void FUNCTION bisect1(FUNCTION f, float a, float b, int m)
float fa,fb,fc,fehler
int n
fa ← f(a)
fb ← f(b)
IF sgn(fa)=sgn(fb)
  THEN
    output "Fehler: f(a) und f(b) haben dasselbe Vorzeichen"
    output a, b, fa, fb
  ELSE
    fehler ← b-a
    FOR n=1 TO m DO
      fehler ← fehler / 2
      c ← a + fehler
      fc ← f(c)
      IF sgn(fa) ≠ sgn(fc)
        THEN
          b ← c
          fb ← fc
        ELSE
          a ← c
          fa ← fc
      END IF
      output n, fc, error
    END DO
END IF
END FUNCTION bisect1
```

Wir codieren nun diesen Entwurf in C. Um die ganze Sache testbar zu gestalten sind neben der eigentlichen Funktion zur Bestimmung der Nullstelle auch ein Hauptprogramm vorgesehen, dass die Eingabe der Intervallgrenzen erlaubt und das Verfahren aufruft.

```
/* BISECT1:
   Nullstellenbestimmung mit der Methode der Intervallhalbierung.
   Implementierung des zweiten Entwurfes anhand des Pseudocodes.

   Arbeitsweise:
   Die Funktion bisect1(f,a,b,m) sucht in m Durchläufen die Nullstelle
   von f im Intervall [a,b] und gibt bei jedem Durchlauf die Werte des
   aktuellen Suchintervalles, den Funktionswert sowie den Fehlerbereich an.

   Autor: Gerhard Krucker
   Datum: 6.2.1995
   Sprache: MS-Visual C++ (QuickWin)
*/
#include <stdio.h>
#include <math.h>
#define sign(x) (((x)<0)?-1:1)

/* Funktion, von der die Nullstelle zu bestimmen ist:
    x^7 + sin(x) -18.5 */
float f(float x)
{ float y;

  y =(float)(pow(x,7.0) + sin(x) - 18.5);
  return y;
}

void bisect1(float (*f)(float),float a, float b, int m)
{  int n;
   float fa,fb,fc,c;
   float fehler;

   fa = f(a);
   fb = f(b);
   if (sign(fa) == sign(fb))
       printf("Fehler: f(a) und f(b) haben dasselbe Vorzeichen\n"
              "a=%f\tb=%f\tf(a)=%f\tf(b)=%f\n",a,b,fa,fb);
      else
        { fehler = b - a;
          for (n=0; n < m; n++)
            { fehler /= 2;
              c = a + fehler;
              fc = f(c);
              if (sign(fa) != sign(fc))
                 { b = c;
                   fb = fc;
                 }
                else
                 { a = c;
                   fa = fc;
                 }
               printf("Durchlauf %d: c=%f\tf(c)=%f\tmax. fehler=%f\n",n,c,fc,fehler);
              }
        }
   }

main()
{  int m;      /* Anzahl auszufuehrender Iterationsschritte */
   float a,b;  /* Randwerte des Suchintervalles als Startvorgabe */

   printf("\nGeben Sie die Randwerte a, b des Suchintervalles ein: ");
   scanf("%f %f",&a, &b);
   printf("Geben Sie die Anzahl auszuführender Iterationen ein: ");
   scanf("%d",&m);

   bisect1(f,a,b,m);
  return 0;
}
```

Ein Testdurchlauf für $x^7 + \sin(x) - 18.5 == 0$ liefert die Nullstelle im Suchintervall [1,2]:

```
Geben Sie die Randwerte a, b des Suchintervalles ein: 1 2
Geben Sie die Anzahl auszuführender Iterationen ein: 10
Durchlauf 0: c=1.500000 f(c)=-0.416568  max. fehler=0.500000
Durchlauf 1: c=1.750000 f(c)=32.749062  max. fehler=0.250000
Durchlauf 2: c=1.625000 f(c)=12.419355  max. fehler=0.125000
Durchlauf 3: c=1.562500 f(c)=5.237333   max. fehler=0.062500
```

```
Durchlauf 4: c=1.531250 f(c)=2.238107   max. fehler=0.031250
Durchlauf 5: c=1.515625 f(c)=0.869881   max. fehler=0.015625
Durchlauf 6: c=1.507813 f(c)=0.216698   max. fehler=0.007813
Durchlauf 7: c=1.503906 f(c)=-0.102393  max. fehler=0.003906
Durchlauf 8: c=1.505859 f(c)=0.056534   max. fehler=0.001953
Durchlauf 9: c=1.504883 f(c)=-0.023083  max. fehler=0.000977
```

Dieses Verfahren zeigt, dass bei geschickter Wahl der Randwerte die Nullstelle in kurzer Zeit, d.h. in wenigen Iterationsschritten in einer guten Näherung bestimmt wird.

Eleganter wäre aber ein Verfahren wo wir einfach eine Fehlerschranke ε vorgeben können und das Verfahren solange rechnet bis die gewünschte Genauigkeit erreicht wird, oder eine bestimmte Anzahl Durchläufe überschritten wird. Das Setzen einer oberen Schranke der Anzahl Durchläufe ist sinnvoll, da manchmal aufgrund ungeschickter Wahl der Grenzen keine Konvergenz erreicht wird. (Dies kann dann der Fall sein, wenn mehrere Nullstellen im Suchintervall liegen.)

Wir machen hierzu eine Änderung im Entwurf: Die geforderte Fehlerschranke wird der Funktion als Argument übergeben. Sobald diese Fehlerschranke unterboten wird, wird die Iteration abgebrochen.

```
void FUNCTION bisect2(FUNCTION f, float a, float b, float epsilon, int m)
float fa,fb,fc
int n
fa ← f(a)
fb ← f(b)
IF sgn(fa)=sgn(fb)
  THEN
    output "Fehler: f(a) und f(b) haben dasselbe Vorzeichen"
    output a, b, fa, fb
  ELSE
    n = 0;
    DO
      c ← (b - a)/2
      fc ← f(c)
      IF sgn(fa) ≠ sgn(fc)
        THEN
          b ←c
          fb ← fc
        ELSE
          a ←c
          fa ← fc
      END IF
    WHILE (n <m) AND (abs(b-a) > epsilon)
    END IF
END FUNCTION bisect2
```

Eine mögliche Realisierung in C für dieses Verfahren wäre:

```
/* BISECT2:
   Nullstellenbestimmung mit der Methode der Intervallhalbierung.
   Implementierung des dritten Entwurfes anhand des Pseudocodes.

   Arbeitsweise:
   Die Funktion bisect2(f,a,b,epsilon,m) bestimmt die Nullstelle von f im Intervall [a,b].
   Das Verfahren terminiert sobald die Nullstelle besser als der maximale Fehler epsilon
   bestimmt ist, oder m Durchlaeufe ueberschritten wurde.

   Autor: Gerhard Krucker
   Datum: 6.2.1995
   Sprache: MS-Visual C++ (QuickWin)
*/

#include <stdio.h>
#include <math.h>

#define sign(x) (((x)<0)?-1:1)

/* Funktion, von der die Nullstelle zu bestimmen ist:
    x^7 + sin(x) -18.5 */
float f(float x)
{ float y;

  y =(float)(pow(x,7.0) + sin(x) - 18.5);
  return y;
}

void bisect2(float (*f)(float),float a, float b, float epsilon, int m)
{  int n;
   float fa,fb,fc,c;

   fa = f(a);
   fb = f(b);
   if (sign(fa) == sign(fb))
       printf("Fehler: f(a) und f(b) haben dasselbe Vorzeichen\n"
              "a=%g\tb=%g\tf(a)=%g\tf(b)=%g\n",a,b,fa,fb);
      else
        { n = 0;
          do {
            n++;
            c = (a + b) /2;
            fc = f(c);
            if (sign(fa) != sign(fc))
               { b = c;
                 fb = fc;
               }
             else
               { a = c;
                 fa = fc;
               }
            } while (n < m && fabs(b-a) > epsilon);
          if (n >= m)
              printf("Verfahren erreichte die maximale Anzahl Iterationen (%d) und hatte \n"
                     "keine Loesung mit dieser Fehlerschranke gefunden!\n"
                     "x=%g\tf(x)=%g\tfehler=%g\n",n,c,fc,b-a);
             else printf("Nullstelle: x=    f(x)=%f\n",c,fc);
       }
}

main()
{  int max=100;       /* Maximale Anzahl auszufuehrender Iterationsschritte */
   float a,b;         /* Randwerte des Suchintervalles als Startvorgabe */
   float epsilon;     /* Genauigkeit der Nullstelle */
   printf("\nGeben Sie die Randwerte a, b des Suchintervalles ein: ");
   scanf("%f %f",&a, &b);
   printf("Geben Sie die gewuenschte Genauigkeit ein: ");
   scanf("%f",&epsilon);

   bisect2(f,a,b,epsilon,max);
   return 0;
}
```

Ein Testlauf für dieses Programm liefert:

```
Geben Sie die Randwerte a, b des Suchintervalles ein: -10 10
Geben Sie die gewuenschte Genauigkeit ein: 1E-5
Nullstelle: x=  f(x)=1.505166
```

Dieses Programm bestimmt also mit beliebiger Genauigkeit eine Nullstelle von *f*.

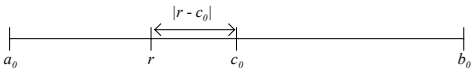
Beliebige Genauigkeit heisst hier, dass die Genauigkeit bis zu einer bestimmten unteren Grenze bestimmt werden kann. Für float's ist dies in diesem Verfahren ca. $3.5 \cdot 10^{-6}$. Die untere Grenze ist durch die Rechnung fabs() und den Vergleich gegeben.

5.1.1.4 Konvergenzverhalten der Methode der Intervallhalbierung

Nachfolgend eine Betrachtung zur Genauigkeit der Näherung und dem Konvergenzverhalten der Methode:

Ist *f(x)* eine stetige Funktion, die in einem Intervall $[a_0,b_0]$ ihr Vorzeichen wechselt, dann gilt nach Satz: Die Funktion *f(x)* hat mindestens eine Nullstelle im Intervall $[a_0,b_0]$. Ferner können wir den maximalen Fehler der Näherung für die Nullstelle *r* festlegen, indem wir vom Mittelpunkt des Intervalls $c_0=(a_0+b_0)/2$ ausgehen. Die maximale Differenz wird:

$$|r - c_0| \le \frac{b_0 - a_0}{2} \tag{5.1}$$



Somit wird der maximale Fehler für den *n*-ten Durchlauf:

$$|r - c_n| \le \frac{b_n - a_n}{2} \qquad (n \ge 0) \tag{5.2}$$

Nun können wir den maximalen Fehler des Verfahrens nach *n* Durchläufen bestimmen:

$$|r - c_n| \le \frac{b_0 - a_0}{2^{n+1}} \qquad (n \ge 0) \tag{5.3}$$

Wir können also folgende Aussage formulieren:
Wenn die Funktion *f(x)* in einem Intervall [*a,b*] stetig ist und die Vorzeichenbedingung $f(a)f(b) < 0$ erfüllt ist, findet die Methode der Intervallhalbierung die Nullstelle nach *n* Durchläufen mit einer Genauigkeit besser als $(b-a)/2^{n+1}$. (Der erste Durchlauf ist Durchlauf #0.)

Wir können nun auch die notwendige Anzahl Durchläufe für einen maximalen Fehler ε berechnen. Wir müssen hierzu die Ungleichung lösen:

$$\frac{b-a}{2^{n+1}} \le \varepsilon \qquad \rightarrow \qquad n > \frac{\ln(b-a) - \ln(\varepsilon)}{\ln 2} - 1 \qquad n\text{: Anzahl Durchläufe} - 1. \tag{5.4}$$

Wir sehen, dass die Anzahl Schritte für eine bestimmte Genauigkeit überhaupt nicht von der zu untersuchenden Funktion *f* abhängt, sondern lediglich von der Intervallbreite.

Beispiel:
Wir suchen eine Nullstelle im Intervall $[a,b]=[1,2]$ und möchten diese mit einer Genauigkeit besser als 10^{-5} bestimmen. Wie viele Durchläufe sind notwendig?

$$n > \frac{\ln(b-a)-\ln(\varepsilon)}{\ln 2} - 1 = \frac{\ln(2-1)-\ln(1\cdot 10^{-5})}{\ln 2} - 1 = \frac{-\ln(1\cdot 10^{-5})}{\ln 2} - 1 = 15.609\ldots \approx 16$$

$\rightarrow$ 17 Durchläufe

5.1.1.5 Rekursive Version der Intervallhalbierung

Eines der wichtigsten Konzepte der Informatik ist die *Rekursion*. Die Rekursion ist einfach gesagt eine 'mächtigere' Form der Iteration. (Genauer: Iteration ist ein Spezialfall der Rekursion). Bei der Rekursion wird folgendes Prinzip verfolgt:

Ein komplexes Problem in ein strukturell gleiches, aber etwas leichter zu lösendes Problem zerlegt. Dieses Verfahren wiederholt man solange bis die Lösung trivial ist. Anschliessend werden alle Teillösungen zur Gesamtlösung zusammengefügt.

Beispiel einer rekursiven Formulierung: Fakultät einer ganzen Zahl n:

$$n! := \begin{cases} 1 & n = 0,1 \\ n(n-1)! & n > 1 \end{cases}$$

Dieses im Prinzip bestechend elegante Verfahren hat doch gewisse Nachteile. Erstens ist diese Methode nicht auf jede Problemstellung anwendbar. Zweitens ist der Aufwand zum Halten der Zwischenresultate beträchtlich, so dass die Effizienz von rekursiven Verfahren oft eher bescheiden ist. Eine rekursive Lösung wählt man zweckmässigerweise dort, wo die Lösung (als Verfahren) bereits rekursiv definiert ist.

Rekursive Lösungen sind meist sehr einfach in der Formulierung, sowohl im Entwurf wie auch in der konkreten Codierung.

Beispiel:

```
unsigned int fak(unsigned int n)
{ if (n==0 || n==1)
     return 1;
   else
     return n*fak(n-1);
}
```

Besonders zu beachten ist, dass die Rekursion in jedem Fall sauber abbricht, da sonst der Stack im Rechner überläuft und der Rechner schlimmstenfalls hängen bleibt.

Wir können nun im entsprechenden Teilintervall wo die Nullstelle liegt einen rekursiven Aufruf zur Verfeinerung der Lösung durchführen. Charakteristisch ist hier, dass die Iterationsschleife wegfällt. Diese ist durch die Rekursion ersetzt worden. Das Abbruchkriterium für die Rekursion ist das Erreichen der geforderten Genauigkeit oder das Überschreiten einer gegebenen maximalen Rekursionstiefe.

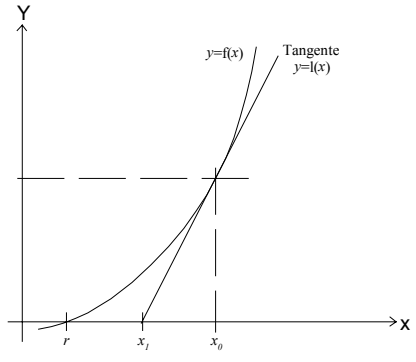
```
int n=0
void FUNCTION bisect_rek(FUNCTION f, float a, float b, float epsilon, int limit)
float fa,fb,fc
IF n > limit
  output "Fehler: Maximale Anzahl Rekursionen ueberschritten!"
  return
END IF
c ← (b - a)/2
fc ← f(c)
IF  abs(fc) <= epsilon
     output(c,fc,n)
     return
  ELSE
    n++
    fa ← f(a)
    fb ← f(b)
    IF sgn(fa) ≠ sgn(fc)
       THEN
         bisect_rek(f,a,c,epsilon,limit)
       ELSE
        bisect_rek(f,c,b,epsilon,limit)
    END IF
END IF
END FUNCTION bisect_rek
```

5.1.2 Verfahren nach Newton

Die Nullstellenbestimmung nach Newton ist ebenfalls ein iteratives Verfahren. Es erlaubt die näherungsweise Bestimmung einer Nullstelle einer Funktion deren Ableitung bekannt ist. Diese Methode konvergiert wesentlich rascher als die Methode der Intervallhalbierung. Meist ist bereits nach 5 Durchläufen eine Genauigkeit besser als 10^{-8} erreicht.

Nachteilig ist, dass sowohl die Funktion selbst wie auch deren Ableitung für die Berechnung vorliegen müssen. Ebenso muss die Funktion nicht nur stetig, sondern auch differenzierbar im ganzen Suchintervall sein. (Bemerkung: Es gibt Funktionen die sind zwar stetig, aber nicht differenzierbar.)

Arbeitsweise:
Von einem gegebenen Näherungswert x_0 wird eine verbesserte Näherung x_1 bestimmt, indem die Tangente an die Funktion an der Stelle $f(x_0)$ gelegt wird und der Schnittpunkt mit der x-Achse bestimmt wird:



Etwas genauer heisst dies:
$f(x)$ ist im ganzen zu betrachtenden Bereich differenzierbar. Daraus folgt, dass wir in jedem Punkt $(x,f(x))$ eine Tangente an die Funktion legen können. Die Tangente lässt sich als lineare Funktion formulieren:

$$l(x) = f'(x_0)(x - x_0) + f(x_0) \tag{5.5}$$

Wir erhalten den Punkt x_1 indem wir den Schnittpunkt mit der x-Achse bestimmen, d.h. $l(x) = 0$ auswerten und erhalten:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{5.6}$$

Der Punkt x_0 ist der Startpunkt für die Nullstellensuche. Er wird vorgegeben als erste Schätzung für die Nullstelle. Im schrittweisen Durchlauf erhalten wir neue, verbesserte Näherungen für die Nullstelle von f:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$
$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

Einen anderen Ansatz, aufbauend auf der Taylorreihenentwicklung ist im Kapitel "Konvergenzverhalten der Newton-Methode" gezeigt.

Beispiel:
Wir bestimmen die Nullstellen nach dem Newton-Verfahren, für die Funktion $f(x) = x^3 - x + 1$ mit dem Schätzwert x_0=1:

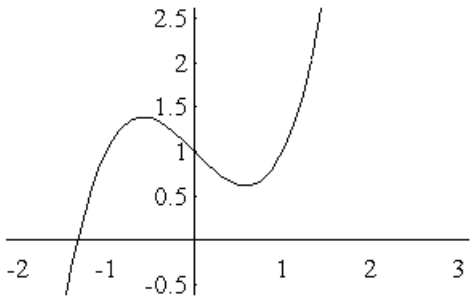
Aus der Definition des Verfahrens bestimmen wir schrittweise:

$$f(x) = x^3 - x + 1 \qquad f'(x) = 3x^2 - 1$$
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1^3 - 1 + 1}{3 \cdot 1^2 - 1} = \frac{1}{2}$$
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = \frac{1}{2} - \frac{\left(\frac{1}{2}\right)^3 - \frac{1}{2} + 1}{3\left(\frac{1}{2}\right)^2 - 1} = 3$$
$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 3 - \frac{3^3 - 3 + 1}{2 \cdot 3^2 - 1} = \frac{53}{26}$$

Wir sehen hier, dass die x_i keine monotone Folge zu einem $x_n \to r$ bilden. Dies ist der Fall, wenn zwischen der realen Nullstelle r und unserem x_i eExtremalstellen, Sattel- oder Wendepunkte liegen.

Für obiges Beispiel kann das Verhalten mit dem Graph veranschaulicht werden:

```
f[x_]:=x^3-x+1
Plot[f[x],{x,-2,3}]
```



5.1.2.1 Entwurf und Codierung

Bestechend ist die Einfachheit des Newton-Verfahrens. Dies zeigt sich bereits im Entwurf mit Pseudocode:

```
void FUNCTION newton(FUNCTION f,FUNCTION f', float x, int m)
int n
float fx

fx ← f(x)
output 0,x,fx
FOR n=1 TO m DO
   x ← x-fx/f'(x)
   fx ← f(x)
   output n,x,fx
END DO
END FUNCTION newton
```

Wir betrachten die Arbeitsweise des Verfahrens, indem wir die einzelnen Schritte beim Lösen der Gleichung $x^3 + 5x^2 = -x + 10$ mitverfolgen. Die Lösung der Gleichung erfolgt durch Suchen der Nullstelle der zusammengesetzten Funktion $f(x) = x^3 + 5x^2 + x^2 - 10$:

```
Geben Sie den Schaetzwert fuer die Nullstelle ein: 2
Geben Sie die Anzahl Durchlaeufe ein: 6
Durchlauf 0:    x=2          f(x)=20
Durchlauf 1:    x=1.39394        f(x)=3.81779
Durchlauf 2:    x=1.21011        f(x)=0.304058
Durchlauf 3:    x=1.19273        f(x)=0.00260179
Durchlauf 4:    x=1.19258        f(x)=1.96391e-007
Durchlauf 5:    x=1.19258        f(x)=1.30885e-015
```

Wir sehen, dass das Verfahren sehr rasch konvergiert. Selbst wenn der Schätzwert recht weit neben der Nullstelle liegt. Hier ist im 5. Durchlauf die Nullstelle bis zur maximal möglichen Rechengenauigkeit der `double`-Codierung bestimmt worden.
Eine konkrete Realisation in C für das Newton Verfahren:

```
/* NEWTON:
   Nullstellenbestimmung nach der Methode NÈWTON-RAPHSON.

   Arbeitsweise:
   Von einem Naeherungswert x0 ausgehend wird eine verbesserte Naeherungx x1 bestimmt, indem
   die Tangente an die Funktion an der Stelle f(x0) gelegt wird und der Schnittpunkt mit x-Achse
   bestimmt wird.
   Die Funktion newton(f,x,m) bestimmt die Nullstelle von f iterativ in m Durchläufen.

   Autor: Gerhard Krucker
   Datum: 6.2.1995
   Sprache: MS-Visual C++ (QuickWin)
*/

#include <stdio.h>
#include <math.h>

/* Funktion, von der die Nullstelle zu bestimmen ist:
    x^3 + 5x^2 + x - 10 */
double f(double x)
{ double y;

  y =x*x*x + 5*x*x + x - 10;
  return y;
}

/* Ableitung f'(x) der Funktion, von der die Nullstelle zu bestimmen ist:
    3x^2 + 10x + 1 */
double df(double x)
{ double y;

  y =3*x*x + 10*x + 1;
  return y;
}
```

```
void newton(double (*f)(double),double (*df)(double), double x, int m)
{  double fx;
   int n;

   fx=f(x);
   printf("Durchlauf %d:\tx=%g\tf(x)=%g\n",0,x,fx);

   for (n=1; n < m;n++)
     { x = x- fx/df(x);
       fx = f(x);
       printf("Durchlauf %d:\tx=%g\tf(x)=%g\n",n,x,fx);
     }
}

main()
{  double x;           /* Schaetzwert fuer die Nullstelle */
   int m;              /* Anzahl Durchlaeufe (Iterationen) */

   printf("\nGeben Sie den Schaetzwert fuer die Nullstelle ein: ");
   scanf("%lf",&x);
   printf("Geben Sie die Anzahl Durchlaeufe ein: ");
   scanf("%d",&m);

   newton(f,df,x,m);

   return 0;
}
```

5.1.2.2 Konvergenzverhalten der Newton-Methode

Bei der praktischen Benutzung der Methode haben wir festgestellt, dass das Newton-Verfahren relativ rasch konvergiert. In der Tat werden in jedem Durchlauf die Anzahl richtiger Stellen etwa verdoppelt. Somit erhalten wir im ersten Durchlauf 0 und in den weiteren Durchläufen 1, 2, 3, 6, 12, 24, ... exakte Stellen für die Nullstelle. Dadurch erreichen wir oft, dass wir nach 5 oder 6 Durchläufen bereits an der Rechengenauigkeit der Maschine angelangt sind. Wir betrachten nun worauf diese rasche Konvergenz begründet ist.

Zuerst leiten wir die Newtonsche Iterationsformel her um $f(r)=0$ zu lösen:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \tag{5.7}$$

Wenn $f(x)$ k-mal im Entwicklungspunkt differenzierbar ist, kann sie durch eine Taylorreihe mit *(k-1)* Gliedern und einem Restglied dargestellt werden.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)(x-x_0)^n}{n!} \qquad \text{resp.} \quad f(x) = \sum_{n=0}^{k} \frac{f^{(n)}(x_0)(x-x_0)^n}{n!} + R(x) \tag{5.8}$$

Das Restglied verkörpert den Fehler der Näherung zum effektiven Wert und kann durch geeignete Formeln bestimmt werden.

Ist unsere Funktion eine 'vernünftige' Funktion (d.h. mindestens 2 mal differenzierbar im ganzen zu betrachtenden Intervall) kann $f(x)$ durch die Taylorreihe mit drei Gliedern dargestellt werden. Wir erhalten eine lineare Funktion und ein quadratisches Restglied.

Wir setzen nun für $f(x=r)=0$ konkret in die Taylorformel ein und beschränken uns auf den linearen Teil der Reihenentwicklung. Wir definieren ferner dass x_n unser r sein soll:

$$f(r) = f(x_{n-1}) + (r - x_{n-1})f'(x_{n-1}) + \frac{1}{2}(r - x_{n-1})^2 f''(\xi) \quad (5.9)$$

$$0 = f(x_{n-1}) + (x_n - x_{n-1})f'(x_{n-1}) \quad (5.10)$$

Wir stellen die Gleichung nach x_n um und erhalten die Iterationsformel nach Newton:

$$r \approx x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \quad (5.11)$$

Da wir aber nur eine endliche Anzahl Durchläufe haben ist x_n nur ein Näherungswert für die Nullstelle *r*.

Für die Konvergenzbetrachtung interessiert uns nun wie gross der Fehler ist. Dazu subtrahieren wir die Gleichungen aus (5.9) , (5.10) und erhalten:

$$0 = (r - x_n)f'(x_{n-1}) + \frac{1}{2}(r - x_{n-1})^2 f''(\xi) \quad (5.12)$$

Wir definieren nun die Abweichung $e_n = |r - x_n|$ und setzen ein:

$$0 = e_n f'(x_{n-1}) + \frac{1}{2}e_{n-1}^2 f''(\xi) \quad (5.13)$$

Unter der Voraussetzung, dass das Verfahren konvergiert, kann man x_{n-1} als auch ξ durch die Lösung *r* ersetzen und erhalten e_n:

$$e_n \approx -\frac{f''(r)}{2f'(r)}e_{n-1}^2 \quad (5.14)$$

Der Fehler nimmt also in jedem Durchlauf etwa zum Quadrat ab. Das bedeutet, dass sich die Zahl der verbesserten Dezimalstellen ungefähr verdoppelt.

5.1.2.3 Probleme beim Newton-Verfahren

Wichtig ist, dass eine gute Näherung für die Nullstelle als Startwert gewählt wird. Besonders kritisch wird es, wenn die Funktion mehrere nahe beieinander liegende Nullstellen hat. Mit einem Graph der Funktion kann man Schätzwerte für die Nullstelle bestimmen.

Ein bis jetzt nicht diskutiertes Problem ist, wenn die Funktion **mehrfache Nullstellen** besitzt. Hier gilt also $f(r)=0$ und $f'(r)=0$. In diesem Fall haben wir keine quadratische Konvergenz mehr, sondern **nur noch lineare Konvergenz.**

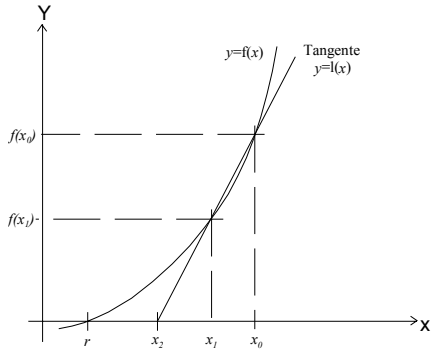
Ebenso haben wir das langsame Hinarbeiten des Verfahrens festgestellt, wenn Extremalwerte der Funktion zwischen Schätzwert und Nullstelle liegen.

5.1.3 Nullstellenbestimmung mit Regula-Falsi (Sekantenmethode)

Die Newton-Methode zeichnet sich durch eine rasche Konvergenz bei einfachen Nullstellen aus und wäre somit ein sehr gutes und universelles Verfahren für die Nullstellenbestimmung. Der grosse Nachteil ist aber, dass auch die Ableitung der Funktion zur Berechnung vorliegen muss.

Die Methode Regula-Falsi arbeitet im Prinzip nach dem Ansatz von Newton, die eine Tangente an den Punkt legt. Die Regula-Falsi umgeht aber die Forderung nach der ersten Ableitung indem man

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{5.15}$$



anstatt einer Tangente eine Sekante an die Funktion legt. Dies entspricht dem Differenzenquotienten als Näherung für die Ableitung:

Wir setzen diesen Differenzenquotienten in die Newton-Iterationsformel ein:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{5.16}$$

Diese Iterationsvorschrift bestimmt jetzt die neue Näherung x_{n+1} ohne die erste Ableitung. Zur Bildung des x_{n+1} sind nun die beiden Vorgängerwerte x_n und x_{n-1} notwendig. Dies hat insbesondere die Konsequenz, dass wir für dieses Verfahren zwei Näherungswerte für den Start benötigen.

Einen Entwurf für die Methode Regula-Falsi führen wir hier mit Nassi-Shneiderman durch:

Funktion $f(x)$ im Quelltext eingeben und Programm kompilieren
Genauigkeit ε einlesen
Startwerte x_0 und x_1 einlesen
$x_0 \leftarrow x_1$
$x_1 \leftarrow x_2$
$x_2 \leftarrow x_1 - f(x_1) * (x_1 - x_0) / (f(x_1) - f(x_0))$
Ausgabe x_2
bis $\lvert x_2 - x_1 \rvert < \varepsilon$

Nachfolgend eine mögliche konkrete Implementierung des Entwurfes in C. Zusätzlich berücksichtigt diese Variante, dass die Iteration nach einer maximalen Anzahl Durchläufen (hier 100) abbricht, auch wenn keine Nullstelle gefunden wurde:

```
/* REGULA FALSI                                                          File:REGULA.C
   Nullstellenbestimmung mit der Methode Regula Falsi.
   Anders als bei der Newton-Methode, wird hier die Ableitung der Funktion
   durch den Differenzenquotienten ersetzt.

   Autor: Gerhard Krucker
   Datum: 9.2.1995
   Sprache: MS-Visual C++ V1.5 (QuickWin)
*/

#include <stdio.h>
#include <math.h>

/* Funktion von der die Nullstelle zu bestimmen ist:
    x^7 + sin(x) -18.5 */
double f(double x)
{ double y;

  y =pow(x,7.0) + sin(x) - 18.5;
  return y;
}

main()
{  int n;               /* Durchlaufzaehler */
   const int max=100;   /* Maximale Anzahl Durchläufe */
   double x0,x1,x2,f0,f1;
   double epsilon;

   printf("Eingabe der Genauigkeit: ");
   scanf("%lf",&epsilon);
   printf("Eingabe der ersten und zweiten Näherung x1 x2: ");
   scanf("%lf %lf",&x1,&x2);


   /* Regula Falsi Iteration */
   n=1;
   f1 = f(x1);

   do {
      x0 = x1;
      x1 = x2;
      f0 = f1;
      f1 = f(x1);
      x2 = x1 - f1 * (x1 -x0) / (f1 - f0);
      n++;
      printf("x=%g\tf(x)=%g\n",x2,f(x2));
     } while ((fabs(x2-x1) > epsilon) && (n < max));

  return 0;
}
```

Ein Testdurchlauf mit $x^7 + \sin(x) - 18.5 == 0$ ergibt:

```
Eingabe der Genauigkeit: 1E-5
Eingabe der ersten und zweiten Näherung x1 x2: 2 3
x=1.94636       f(x)=88.2479
x=1.90166       f(x)=72.382
x=1.69776       f(x)=23.1492
x=1.60189       f(x)=9.56562
x=1.53437       f(x)=2.52181
x=1.5102        f(x)=0.414343
x=1.50545       f(x)=0.0231009
x=1.50517       f(x)=0.00022976
x=1.50517       f(x)=1.29253e-007
```

5.1.3.1 Konvergenzbetrachtung der Methode Regula-Falsi

Der grosse Vorteil der Methode Regula-Falsi ist, dass nur eine einzige Funktion pro Durchlauf ausgewertet werden muss und das Verfahren trotzdem rasch konvergiert. Aufgrund ähnlicher Überlegungen wie bei der Newton-Methode findet man, dass die Fehler wie folgt beschrieben werden können:

$$e_{n+1} = \frac{1}{2}\left(\frac{f''(\xi_n)}{f'(\zeta_n)}\right) e_n\, e_{n-1} \approx \frac{1}{2}\left(\frac{f''(r)}{f'(r)}\right) e_n\, e_{n-1} \tag{5.17}$$

Wobei ξ_n und ζ_n im kleinsten Intervall liegen die *r*, x_n und x_{n-1} enthalten. Somit konvergiert das Verhältnis :

$$\frac{e_{n+1}}{e_n\, e_{n-1}} \quad \xrightarrow[n\to\infty]{} \quad \frac{1}{2}\left(\frac{f''(r)}{f'(r)}\right) \tag{5.18}$$

Die Geschwindigkeit der Konvergenz für Regula-Falsi liegt etwa zwischen der Intervallhalbierung und der Newton-Methode.

5.1.4 Modifiziertes Newton-Verfahren

Eine sehr interessante Verbesserung zum Verfahren von Newton wird in [12] beschrieben. Dabei werden die Vorteile von Regula-Falsi (nur eine Funktion auszuwerten) und Newton (rasche Konvergenz, nur ein Startwert) miteinander vereinigt.

Bei der Methode Regula-Falsi wird die Ableitung durch einen groben Differenzenquotienten der Art

$$f'(x_n) \approx \frac{f(x_n + h) - f(x_n)}{h} \tag{5.19}$$

angenähert, wobei *h* die recht grosse Schrittweite $h = x_{n-1} - x_n$ist. Eine Verbesserung der Methode wäre also den Differenzenquotienten genauer zu berechnen, so dass er näher bei der Ableitung liegt. Dazu müsste man das *h* kleiner wählen:

$$f'(x_n) = \lim_{h\to 0} \frac{f(x_n + h) - f(x_n)}{h} \tag{5.20}$$

Zur Rechnung wählt man das *h* also möglichst klein. Als praktischer Wert kann $h=10^{-8}$ als gute Grösse empfohlen werden. Kleinere Werte bringen nur bedingt bessere Resultate, hingegen treten dann Probleme aufgrund von Stellenauslöschung auf.

Wir definieren unsere neue Iterationsvorschrift für das modifizierte Newton-Verfahren "Newton+":

$$x_{n+1} = x_n - \frac{h \cdot f(x_n)}{f(x_n + h) - f(x_n)}$$

Im Vergleich zur Regula-Falsi benötigt diese Rekursionsformel nur einen Startwert. Die Ablaufstruktur ist immer dieselbe und so kann das vorherige Programm leicht dem neuen Formalismus angepasst werden.

```
/* NEWTON PLUS                                                    File:NEWTONPL.C
   Nullstellenbestimmung nach dem modifizierten Newton Verfahren.
   Die Ableitung der Funktion durch einen Differenzenquotient mit
   konstanter kleiner Schrittweite ersetzt.

   Autor: Gerhard Krucker
   Datum: 9.2.1995
   Sprache: MS-Visual C++ V1.5 (QuickWin)
*/

#include <stdio.h>
#include <math.h>

/* Funktion von der die Nullstelle zu bestimmen ist:
    x^7 + sin(x) -18.5 */
double f(double x)
{ double y;

  y =pow(x,7.0) + sin(x) - 18.5;
  return y;
}

main()
{  int n;               /* Durchlaufzaehler */
   const int max=100;   /* Maximale Anzahl Durchläufe */
   const double h=1E-8;  /* Differenz */
   double x1,x2,fx;
   double epsilon;

   printf("Eingabe der Genauigkeit: ");
   scanf("%lf",&epsilon);
   printf("Eingabe des Startwertes: ");
   scanf("%lf",&x1);


   /* Newton+ Iteration */
   n=1;
   do {
      fx = f(x1);
      x2=x1;
      x1 = x1 - fx * h /(f(x1+h) -fx);
      printf("%d:\tx=%g\tf(x)=%g\n",n,x1,fx);
      n++;
      } while ((fabs(x2-x1) > epsilon) && (n < max));

  return 0;
}
```

Ein Testdurchlauf für das Beispiel $x^7 + \sin(x) - 18.5 == 0$ liefert die Werte:

```
Eingabe der Genauigkeit: 1E-8
Eingabe des Startwertes: 2
1:      x=1.75332       f(x)=110.409
2:      x=1.58884       f(x)=33.4201
3:      x=1.51725       f(x)=8.05938
4:      x=1.50545       f(x)=1.00868
5:      x=1.50517       f(x)=0.0231555
6:      x=1.50517       f(x)=1.30468e-005
7:      x=1.50517       f(x)=4.31621e-012
```

5.1.5 Zusammenfassung

Vier Verfahren zur iterativen Nullstellenbestimmung wurden vorgestellt und diskutiert. Sicher kann man feststellen, dass jede Methode ihre Vor- und Nachteile hat. Es gibt somit keine "beste Methode" zur Nullstellenbestimmung, obwohl für die allgemeine Anwendung sicher das Verfahren Newton+ favorisiert werden kann.

Nachfolgend eine Negativ /Positiv Bewertung der einzelnen Verfahren:

Intervallhalbierung:

Negativ: Das Verfahren konvergiert langsam. Zwei Startwerte, zwischen denen sich die Nullstelle befindet, sind notwendig.

Positiv: Wenn im Suchintervall nur eine Nullstelle liegt konvergiert das Verfahren sicher gegen die Nullstelle.

Newton:

Negativ: Zwei Funktionen notwendig: Grundfunktion sowie erste Ableitung. Langsame Konvergenz bei mehrfachen Nullstellen oder wenn Extremalstellen zwischen Schätzwert und Nullstelle liegen.

Positiv: Sehr rasche Konvergenz bei einfachen Nullstellen und wenn der Schätzwert nahe bei der Nullstelle liegt. Einfacher Algorithmus. Nur ein Startwert notwendig.

Regula-Falsi:

Negativ: Langsamere Konvergenz als beim Newton-Verfahren. Grösserer Rechenaufwand. Zwei Startwerte notwendig.

Positiv: Keine Ableitung der Funktion notwendig.

Newton+:

Negativ: Die Differenz *h* kann kritisch sein. Gleiche Probleme für mehrfache Nullstellen und Extrema wie bei Newton.

Positiv: Sehr rasche Konvergenz. Nur ein Startwert. Keine Ableitung notwendig.

5.1.6 Andere Verfahren

Wie anfangs erwähnt, existieren verschiedenste Verfahren zur Nullstellenbestimmung, welche aber teilweise auf sehr spezielle Funktionen zugeschnitten sind.

Eine andere grundsätzliche Methode, worauf sich eine ganze Anzahl andere Verfahren zur Nullstellenbestimmung begründen, sind so genannte Fixpunktverfahren. Sie beruhen dem Banachschen Fixpunktsatz der (sinngemäss) besagt:

Wenn f eine kontrahierende Abbildung ist, dann gibt es genau ein x mit $f(x)=x$. Dieses x wird Fixpunkt von f genannt. Man konstruiert dann aus der zu untersuchenden Funktion eine Fixpunktgleichung und löst diese iterativ.

Weiter können die Verfahren für Gleichungssysteme mit mehreren Variablen erweitert werden. Das Vorgehen mit dem prinzipiellen Ansatz ist in einer Übungsaufgabe 5.3.23 gezeigt.

Ein Verfahren für Polynome, welches komplexe Nullstellen bestimmen kann, wird im Kapitel „Polynome“ vorgestellt (Verfahren nach Bairstow).

5.2 Aufgaben

Methode der Intervallhalbierung

1. Die Intervallgrenzen seien mit a=0.1 und b =1.0 als gegeben. Wie viele Iterationsdurchläufe n sind notwendig, um die Näherung einer Nullstelle mit einem Fehler kleiner als $0.5 \cdot 10^{-8}$ zu bestimmen?

2. Bestimmen Sie alle Nullstellen der Funktion $f(x) = \cos(x) - \cos(3x)$

3. Bestimmen Sie die Nullstellen der Funktion $\ln\left(\frac{1+x}{1-x^2}\right) = 0$

4. Zeigen Sie grafisch, dass die Gleichung $50\pi + \sin x = 100 \arctan x$ unendlich viele Lösungen hat.

5. Wenn f eine Umkehrfunktion besitzt, kann $f(x)$=0 einfach durch x=f^{-1}(0) bestimmt werden. Ist dieses Verfahren für die Praxis eine echte Alternative zur Nullstellensuche? Wo liegen die Nachteile?
   Hinweis: Zeigen Sie die Problematik mit $\sin x = \frac{1}{\pi}$.

6. Bestimmen Sie den Schnittpunkt der beiden Linien $y = 3x$ und $y = e^x$ indem Sie die Nullstelle(n) der Funktion $e^x - 3x = 0$ auf 4 Dezimalstellen genau bestimmen.

7. Um wie viele Binärstellen in der Mantisse der Nullstelle verbessert sich die Genauigkeit bei jedem Durchlauf der Methode der Intervallhalbierung?

   Wie viele Durchläufe sind pro Dezimalstelle Genauigkeit notwendig?

Programmieraufgaben

8. Bestimmen Sie die Nullstellen mit der Methode der Intervallhalbierung für folgende Funktionen im vorgegebenen Intervall mit einer Genauigkeit von $1 \cdot 10^{-10}$:

   a.) $f(x) = x^3 + 3x - 1$ in $[0,1]$
   b.) $g(x) = x^3 - 2\sin x$ in $[0.5,2]$
   c.) $h(x) = x + 10 - x\cosh\left(\frac{50}{x}\right)$ in $[120,130]$

9. Verfeinern Sie Ihr Nullstellenprogramm so, dass es (fast) alle reellen Nullstellen findet. Ein Ansatz wäre in einer bestimmten Schrittweite die Funktion zu prüfen, ob ein Vorzeichenwechsel der Funktionswerte stattfindet und diese näher zu untersuchen. Die Suchschrittweite sowie Randwerte sollten manuell eingegeben werden können.

10. Wilkinson zeigte 1963 mit nachfolgendem Polynom, dass kleinste Änderungen an den Koeffizienten eines Polynoms massive Auswirkungen auf die Nullstellen haben. Das hierzu verwendete Wilkinson-Polynom hat die Gestalt:

$$p(x) = (x-1)(x-2)\cdots(x-20)$$

Es besitzt wie man leicht ersieht, 20 Nullstellen 1,2,3,...,20. Sie haben nachfolgend die Koeffizienten der Einzelglieder von Mathematica ausmultipliziert dargestellt.

$a_0 = 2432902008176640000$	$a_8 = 63030812099294896$	$a_{16} = 53327946$
$a_1 = -8752948036761600000$	$a_9 = -10142299865511450$	$a_{17} = -1256850$
$a_2 = 13803759753640704000$	$a_{10} = 1307535010540395$	$a_{18} = 20615$
$a_3 = -12870931245150988800$	$a_{11} = -135585182899530$	$a_{19} = -210$
$a_4 = 8037811822645051776$	$a_{12} = 11310276995381$	$a_{20} = 1$
$a_5 = -3599979517947607200$	$a_{13} = -756111184500$	
$a_6 = 1206647803780373360$	$a_{14} = 40171771630$	
$a_7 = -311333643161390640$	$a_{15} = -1672280820$	

Versuchen Sie mit Ihrem Programm 20 Nullstellen des Polynoms numerisch zu bestimmen (ε besser als 10^{-3}).

Hinweise benutzen Sie zur Rechnung Gleitkommazahlen vom Typ `double`. Zur Auswertung des Polynoms verwendet man zweckmässigerweise das Verfahren von Horner. Dieses vermeidet weitmöglichst die Stellenauslöschung:

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = \Big(\big((a_n x + a_{n-1})x + \cdots\big)x + a_1\Big)x + a_0$$

11. Bestimmen Sie den Schnittpunkt der Kurven $y = x^3 - 2x + 1$ und $y = x^2$.

12. Programmieren Sie die rekursive Lösung des Verfahrens der Intervallhalbierung.

Verfahren nach Newton

13. Zeigen Sie, dass unter Verwendung des Newton-Verfahrens die Quadratwurzel aus *R* bestimmt werden kann (durch Lösen der Gleichung $x^2=R$). Die Iteration ist dabei definiert durch $x_{n+1}=(x_n+R/x_n)/2$.

14. Zwei der vier Nullstellen von $x^4 + 2x^3 - 7x^2 + 3$sind positiv. Finden Sie diese mit der Newton-Methode auf zwei Dezimalstellen genau.

15. Welche lineare Funktion $ax+b$ Nähert $f(x)=\sin x$ im Punkt $x=\pi/4$ an? In welchem Zusammenhang steht dies zur Newton-Methode?

16. Die Taylorreihenentwicklung für eine Funktion f lässt sich beschreiben:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \cdots$$

Nehmen Sie an, dass $f(x)$, $f'(x)$ und $f''(x)$ einfach berechnet werden kann. Entwickeln Sie nun ein Iterationsverfahren ähnlich der Newton-Methode, das aber drei Terme der Taylorreihe benutzt. Das Verfahren sollte in einem Durchlauf ausgehend von einem Startwert einen verbesserten Wert für die Nullstelle liefern. Zeigen Sie dann, dass das Verfahren kubisch konvergiert.

17. Betrachten Sie wie schwerfällig das Newton-Verfahren auf einem Rechner für $f(x) = (x-1)^m$ mit m=8 oder 12 konvergiert. Erklären Sie das Verhalten mit der Theorie! (Benutzen Sie x_0=1.1)

18. Jede der nachfolgenden Funktionen hat $\sqrt[3]{R}$ als Nullstelle für jede beliebige positive reelle Zahl R. Bestimmen Sie für jede Funktion die Iterationsformeln der Newton-Methode und die Einschränkungen für die Wahl des Startwertes x_0!

a.) $a(x) = x^3 - R$

b.) $b(x) = \frac{1}{x^3} - \frac{1}{R}$

c.) $c(x) = x^2 - \frac{R}{x}$

d.) $d(x) = x - \frac{R}{x^2}$

e.) $e(x) = 1 - \frac{R}{x^3}$

f.) $f(x) = \frac{1}{x} - \frac{x^2}{R}$

g.) $g(x) = \frac{1}{x^2} - \frac{x}{R}$

h.) $h(x) = 1 - \frac{x^3}{R}$

19. Bestimmen Sie die Nullstelle von $f(x) = e^{-x} - \cos x$ die am nächsten bei $\pi/2$ ist.

Programmieraufgaben

20. Entwickeln Sie eine einfache Prozedur (Funktion) `newton`, die $x^3 + 2x^2 + 10x = 20$ mit dem Startwert x_0=2. Bestimmen Sie $f(x)$ und $f'(x)$ durch das Verfahren von Horner. Die Nullstelle soll ausgegeben werden wenn sie genauer als $0.5 \cdot 10^{-5}$ bestimmt worden ist. Alle Zwischenwerte der Iterationdurchläufe sind mit den verbesserten Näherungen und zugehörigen Funktionswerten auszugeben. Setzen Sie zusätzlich für das Verfahren eine Obergrenze von maximal 10 Iterationen.

21. Bestimmen Sie die Lösung der Gleichung

$$2x(1-x2+x)\ln x = x^2 - 1$$

im Intervall [0,1] mit der Newton-Methode unter Verwendung von doppelt genauen Gleitkommazahlen (double). Erstellen Sie eine Tabelle die für jeden Durchlauf die Anzahl der richtig bestimmten Stellen zeigt.

22. Lösen Sie nachfolgendes nichtlineares Gleichungssystem. Hinweis: Eliminieren Sie zuerst y und lösen Sie nachher mit Newton nach x auf. Wählen Sie einen Startwert x_0=1.0.

$x^3 - 2xy + y^7 - 4x^3y = 5$

$y\sin x + 3x^2 y + \tan x = 4$

1. Das Newton-Verfahren kann auch für nichtlineare Gleichungssysteme (simultane Gleichungen) verwendet werden. Die Grundlage hierfür ist eine Taylorreihenentwicklung mit mehreren Variablen. Ein Spezialfall wäre die Nullstelle einer komplexe Funktion: $f(z) = g(z) + jh(z)$wobei *f(z)* eine analytische Funktion der komplexen Variable *z=x+jy* ist (*x* und *y* sind reell) und *g(z)* und *h(z)* sind reelle Funktionen für alle *z*. Die Ableitung ist aufgrund der Cauchy-Riemann Gleichungen:

   $f'(z) = g_x + jh_x = h_y - jg_y$
   $g_x = h_y \qquad h_x = -g_y$ wobei: $g_x = \frac{\partial g}{\partial x} \qquad g_y = \frac{\partial g}{\partial y}$ etc.

   Nun können wir das Iterationsverfahren nach Newton $z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$ ausdrücken:

   $x_{n+1} = x_n - \frac{gh_y - hg_y}{g_x h_y - g_y h_x} \qquad y_{n+1} = y_n - \frac{hg_x - gh_x}{g_x h_y - g_y h_x}$

2. Entwickeln Sie ein Verfahren nach Newton welches auch komplexe Lösungen bestimmen kann. Testen Sie Ihr Verfahren an:

   a.) $z^3 - z - 1 = 0$
   b.) $z^4 - 2z^3 - jz^2 + 4jz = 0$
   c.) $2z^3 - 6(1+j)z^2 - 6(1-j) = 0$
   d.) $z = e^z$

   Hinweis: Benutzen Sie in d.) die Eulersche Identität $e^{jy} = \cos y + j\sin y$.