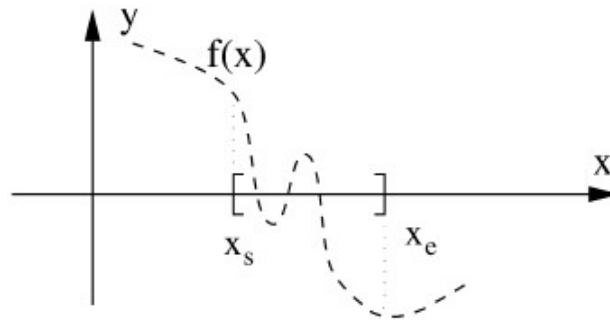


Erklärung zum Programm "Nullstelle"

Zu erst einmal die offizielle Aufgabenstellung:

Ein einfaches Verfahren zur Nullstellensuche bei einer stetigen Funktion $f(x)$ ist das Verfahren der *fortgesetzten Intervallhalbierung*.



1. Schreiben sie eine C-Funktion `float nullstelle(float x_s, float x_e)`, welche für eine gegebene Funktion `float f(float x)` den x-Wert einer Nullstelle zurückgibt. `x_s` und `x_e` beschreiben dabei das Suchintervall. Beim Aufruf kann davon ausgegangen werden, daß $f(x_s)$ und $f(x_e)$ unterschiedliche Vorzeichen haben.

Der Algorithmus soll abbrechen, wenn die Differenz zwischen `x_s` und `x_e` kleiner als eine vorgegebene Toleranzgrenze `epsilon` ist.

Überlegungen zu dieser Aufgabe:

Wir dürfen also davon ausgehen, dass $f(x_s)$, also der zu `x_s` zugehörige y-Wert, ein anderes Vorzeichen hat als der zum `x_e` zugehörige y-Wert. Wenn das so ist, dann muss dazwischen ja irgendwo die Nullstelle sein.

Nun halbieren wir das Intervall und nennen die Mitte des Intervalls `x_m`. Dann schauen wir uns eine der beiden Hälften an. Es ist uns überlassen, welche wir uns anschauen. Schauen wir uns mal die linke Hälfte an.

Wir schauen jetzt also, ob $f(x_s)$ und $f(x_m)$, also der zu `x_m` zugehörige y-Wert, gleiche oder unterschiedliche Vorzeichen haben.

Stellen wir uns vor, sie haben das gleiche Vorzeichen:

Für diesen Fall können wir davon ausgehen, dass sich dazwischen KEINE NULLSTELLE befindet. Demnach müsste sich dann in der rechten Hälfte die Nullstelle befinden.

Für den Fall, dass sie unterschiedliche Vorzeichen haben, muss dazwischen ja irgendwo die Nullstelle liegen.

Sollte dem so sein, dann können wir die rechte Grenze (also das bisherige `x_e`) auf den Wert von `x_m` verschieben und somit die komplette rechte Hälfte des bisherigen Intervalls abschneiden.

Ansonsten, also wenn die Nullstelle in der rechten Hälfte liegen muss, müssten wir die bisherige linke Hälfte, also `x_s` in die Mitte verschieben und in der rechten Hälfte weiter suchen.

Im Klartext: Wir halbieren das Intervall immer wieder und gucken jedesmal, in welcher der beiden Hälften die Nullstelle liegen muss. Das Ganze machen wir so lange, bis `x_s` und `x_e` sehr nah beieinander liegen, dann haben wir uns sehr gut an die Nullstelle angenähert...

Jetzt die offizielle Lösung zu dieser Aufgabe:

```
float f(float x)
{
    return 2*x - 4.0;
}

float nullstelle(float x_s, float x_e)
{
    float epsilon = 0.00001;
    float x_m;

    while((x_e-x_s) > epsilon)
    {

        x_m = (x_s+x_e)/2;

        if(f(x_s) * f(x_m) > 0) // Wenn beide das gleiche Vorzeichen haben
        {
            x_s = x_m;
        }
        else
        {
            x_e = x_m;
        }
    }
    return x_m;
}

main()
{
    cout << "Linke Intervallgrenze: ";
    float x_s;
    cin >> x_s;

    cout << "Rechte Intervallgrenze: ";
    float x_e;
    cin >> x_e;

    float erg = nullstelle(x_s, x_e);

    cout << endl;

    cout << "Nullstelle bei der Funktion 2x-4: " << erg << endl;

    getch();
}
```

Und jetzt gehen wir das Ganze mal Schritt für Schritt durch:

Wir beginnen mit der main-Funktion.

Zuerst fragen wir den Benutzer, in welchem Intervall gesucht werden soll. Dazu fragen zu erst nach der linken Intervallgrenze:

```
cout << "Linke Intervallgrenze: ";
```

Das, was der Benutzer jetzt eingeben soll, muss ja irgendwo gespeichert werden. Noch haben wir keine Variable, in der wir die Eingabe des Benutzers speichern, also „deklarieren“ wir erstmal eine. Sie soll auch Kommawerte annehmen können, also ist sie vom Typ float (wahlweise auch double, aber die Aufgabenstellung hat ja von float-Werten gesprochen).

```
float x_s;
```

Nun haben wir also eine Variable, in der die Eingabe gespeichert werden kann. Also lassen wir den Benutzer jetzt die Eingabe machen und speichern diese in x_s (das s bei x_s steht für Start des Intervalls, ist deswegen die linke Intervallgrenze, das e bei x_e steht für Ende des Intervalls).

```
cin >> x_s;
```

Genau so wie mit der linken verfahren wir nun auch mit der rechten Intervallgrenze:

```
cout << "Rechte Intervallgrenze: ";
```

```
float x_e;
```

```
cin >> x_e;
```

Jetzt haben wir alles, was wir für dieses Programm brauchen. Wir haben das Intervall, in dem gesucht werden soll – mehr brauchen wir nicht, um die Funktion starten zu können, denn die erwartet ja nur eben diese beiden Werte.

Eines bleibt aber noch: Das Ergebnis, das uns die Funktion liefert, also der x-Wert der Nullstelle, wird ja genau an die Stelle zurückgegeben, wo die Funktion aufgerufen wurde. Dieses Ergebnis brauchen wir ja noch, wir müssen es also irgendwo speichern. Deswegen legen wir noch eine Variable vom Typ float an, in der das dann gespeichert wird. Die entsprechende Zeile sieht dann so aus:

```
float erg = nullstelle(x_s, x_e);
```

Dann lassen wir (aus rein ästhetischen Gründen) eine Zeile frei.

```
cout << endl;
```

Das Ergebnis, das uns die Funktion float nullstelle(float x_s, float x_e) geliefert hat, muss jetzt nur noch auf dem Bildschirm ausgegeben werden. Und Ausgaben machen wir ja immer mit cout, also:

```
cout << "Nullstelle bei der Funktion  $2x-4$ : " << erg << endl;
```

Für die Funktion $2x-4$ haben wir uns einfach mal entschieden. Die können wir ja jederzeit ändern (und zwar in der Funktion float f(float x)).

Und damit das Programm nicht sofort beendet, lassen wir es noch auf ein Zeichen warten:

```
getch();
```

So weit die Erklärung der main-Funktion.

Jetzt geht's ans Eingemachte:

Die Funktion

```
float nullstelle(float x_s, float x_e)
```

Jede Funktion hat ja einen "Rückgabety". Was bitte schön gibt sie denn zurück?

Es kommt drauf an.

Jede Funktion, die etwas berechnet und ein Ergebnis liefern soll, muss genau dieses Ergebnis eben zurück geben.

Das wäre wie wenn ich jemanden bitten würde, für mich etwas zu berechnen. Das, was dieser jemand berechnet hat, soll er mir mitteilen, also soll er mir das Ergebnis zurückgeben.

Was ist aber mit einer Funktion, die lediglich etwas auf dem Bildschirm ausgeben soll, und somit keinen Wert zurückgibt?

Nehmen wir als Beispiel eine Funktion, die lediglich alle Werte in einem Array auf die Konsole ausgeben (also lediglich hinschreiben) soll. Diese Funktion schreibt lediglich etwas, gibt aber nichts an die Stelle, wo sie aufgerufen wurde, zurück. Das heißt, ihr Rückgabety ist leer. Also **void** (englisch für „leer“).

Da jetzt float vor der Funktion steht, wissen wir, dass diese Funktion einen Wert vom Typ float zurückgibt. Als Nächstes stehen in der Klammer zwei sogenannte „Parameter“. Klare Sache. Wenn ich jemanden bitte: „Addiere mal bitte zwei Zahlen für mich“. Dann würde dieser jemand als erstes sagen „Geht klar. Welche Zahlen denn?“. Ich müsste ihm also die zwei Zahlen, um die es geht, nennen.

Und genau das macht die Funktion nullstelle. Sie stellt in den zwei Klammern klar, dass sie, um eine Nullstelle in einem bestimmten Intervall zu berechnen, zwei Werte braucht, nämlich den Anfang und das Ende des Intervalls.

In der Aufgabenstellung war von einer „vorgegebenen Toleranzgrenze“ die Rede. Diese Toleranzgrenze sollte natürlich möglichst klein sein, denn um so genauer ist das Ergebnis. Wir legen also die Variable epsilon an und geben ihr einen kleinen Wert.

```
float epsilon = 0.00001;
```

Außerdem machen wir dem Programm schon mal die Variable x_m bekannt. In ihr soll jeweils der Mittelwert des Intervalls gespeichert werden.

```
float x_m;
```

Anschließend machen wir die ständige Intervallhalbierung und nähern uns so der Nullstelle. Das ganze machen wir so lange, bis die Differenz zwischen x_e und x_s (die sich ja aus $x_e - x_s$ errechnen lässt) kleiner als die vorgegebene Toleranzgrenze epsilon ist. Also...

```
while((x_e-x_s) > epsilon)
```

Dann kommt die Klammer...

```
{
```

Jetzt wird der vorher angelegten Variable x_m der Mittelwert des Intervalls zugeordnet. Der Mittelwert ist ja die Addition der Werte geteilt durch die Anzahl der Summanden, also hier x_s und x_e addieren und durch 2 teilen...

$$x_m = (x_s + x_e) / 2;$$

Wenn wir das Intervall halbiert haben, so war der Plan, wollten wir schauen, ob in der linken oder rechten Hälfte die Nullstelle sein muss. Wie das nochmal von Statten gehen sollte – siehe oben.

Wir prüfen also, ob die y-Werte von x_s und x_m (also $f(x_s)$ und $f(x_m)$) das gleiche Vorzeichen haben oder nicht.

Wie prüft man das?

Eine nicht verkehrte Möglichkeit wäre folgende:

if($f(x_s) > 0 \ \&\& \ f(x_m) > 0 \ || \ f(x_s) < 0 \ \&\& \ f(x_m) < 0$)

Also: Wenn $f(x_s)$ größer als 0 **und** $f(x_m)$ größer als 0 **oder** $f(x_s)$ kleiner als 0 **und** $f(x_m)$ kleiner als 0 ist...

Doch da bietet der Mathematiker mal wieder eine elegantere Lösung:

Wenn man die y-Werte miteinander multipliziert, dann ergibt sich Folgendes:

Bei...

...gleichen Vorzeichen: Ergebnis definitiv positiv (denn auch minus mal minus ist plus)

...verschiedenen Vorzeichen: Ergebnis definitiv negativ

Also:

if($f(x_s) * f(x_m) > 0$)

Übrigens: Bei $f(x_s)$ und bei $f(x_m)$ wird jeweils die Funktion **float f(float x)** aufgerufen, die den jeweiligen y-Wert liefert...

Jetzt heißt diese Zeile also:

Wenn die beiden y-Werte das gleiche Vorzeichen haben...

Was machen wir dann?

Wenn sie das gleiche Vorzeichen haben, dann gibt es in diesem Intervall KEINE NULLSTELLE.

Also müssen wir rechts weiter suchen.

Das heißt, wir verschieben die linke Grenze in die Mitte und werden die gleiche Prozedur mit dem nun veränderten Intervall erneut durchführen.

Also bekommt der x_s -Wert den Wert von x_m zugewiesen.

```
{
    x_s = x_m;
}
```

Was machen wir, wenn die y-Werte, die wir betrachtet haben, NICHT das gleiche Vorzeichen haben? Dann verschieben wir die rechte Grenze in die Mitte, also bekommt x_e den Wert von x_m .

```
else
{
    x_e = x_m;
}
}
```

Bei jedem Durchgang ändert sich also entweder die Position von x_s oder die von x_e . Somit wird das Intervall selbst immer kleiner, bis es schließlich irgendwann kleiner als die vorgegebene Toleranzgrenze ist. Die while-Schleife wird verlassen, sobald dies der Fall ist.

Nun haben wir uns also der Nullstelle ziemlich gut genähert (bei $\epsilon = 0.00001$ sind wir also auf eine zehntausendstel Stelle genau – das müsste reichen :-)).

Dieses Ergebnis, also den rausgefundenen x-Wert der Nullstelle, soll uns die Funktion jetzt zurückgeben.

Also:

```
return x_m;
```

Und Klammer zu :-)

```
}
```

Bei der Funktion $2x-4$ muss uns das Programm als nullstelle die 2 ausgeben...

Und das tut das Programm auch...