Auswertung von Messwerten

Eine praxisorientierte Einführung

Jürgen Plate, 10. Oktober 2016

Inhaltsverzeichnis

1

Datenauswertung

Dieses Kapitel soll in die Auswertung bzw. die Visualisierung der mit Sensoren und A/D-Wandler erfassten Daten einführen. Dabei kommen einige statistische Methoden zum Einsatz, die überwiegend der deskriptiven Statistik und Zeitreihenanalyse entstammen. Häufig genügen ganz einfache statistische Verfahren oder sogar nur die grafische Darstellung der erfassten Werte. Deshalb gehe ich nur auf einige grundlegende Methoden ein und bespreche anschließend höchst hilfreiche Tools zur Darstellung.

1.1 Messfehler

Jede Messung ist fehlerbehaftet. Durch viele verschiedene Ursachen wird die zu messende Größe nicht korrekt erfasst. Die Abweichung eines aus Messungen gewonnenen Wertes vom wahren Wert der Messgröße wird „Messabweichung“ oder „Messfehler“ genannt. Sie sollten sich daher bei der Angabe eines Messwertes immer fragen:

- Wie weit kann ich mich auf den angezeigten (ermittelten) Wert als korrekte Aussage über die zu messende Größe verlassen?
- Wie weit kann ich mich auf den festgestellten Zahlenwert verlassen? Es ist beispielsweise sinnlos, eine Temperatur mit drei Stellen hinter dem Komma anzugeben, wenn der Sensor nur auf Zehntelgrad genau messen kann.

Man unterscheidet den **absoluten** und den **relativen** Fehler eines Messwerts. Der absolute Fehler F hat einen Betrag, ein Vorzeichen und dieselbe Einheit wie die Messgröße. Für den angezeigten Wert x_a und den richtigen Wert x_r gilt:

$$F = x_a - x_r \tag{1.1}$$

Der relative Fehler f hat keine Einheit; er kann positiv oder negativ sein:

$$f = \frac{F}{x_r} = \frac{x_a - x_r}{x_r} \tag{1.2}$$

Für prozentuale Angaben wird f noch mit 100 % multipliziert. Beispiel: Gemessen wird eine Spannung von 5,1 V. Tatsächlich beträgt die Spannung 4,9 V. Dann gilt $F = 5,1 - 4,9 = 0,2V$ und $f = (5,1 - 4,9)/4,9 = 0,041$ oder 4,1 %.

Als Quelle für Messfehler kommen in Frage:

- *Gerätefehler* als Folge der Konstruktion, Fertigung, Justierung
- *durch das Messverfahren bedingte Einflüsse* infolge Einwirkung der Messeinrichtung auf die Messgröße (z. B. Belastung eines Spannungsteilers durch den Innenwiderstand der Messeinrichtung)

- *Umwelteinflüsse* als Folge von Umgebungseinflüssen
- *Instabilitäten* des Wertes der Messgröße

Alle Fehler, die in eine Richtung weisen, nennt man „systematische Messabweichungen“ oder „systematische Messunsicherheit U_s“. Systematische Messabweichungen haben Betrag und Vorzeichen. Bekannte systematische Abweichungen (z. B. eine Offsetspannung eines OPV) kann man berichtigen – entweder in der Messanordnung oder rechnerisch im Computer.

Nicht beherrschbare und ungerichtete Abweichungen sind zufällige Messabweichungen. Selbst bei Wiederholung der Messung streuen die Messwerte. Solch zufällige Abweichungen schwanken nach Betrag und Vorzeichen. Anhand einer Fehlerrechnung kann aus der Gesamtheit der Werte ein Mittelwert M und die Messunsicherheit u_z berechnet werden. Der wahre Wert W liegt mit statistischer Wahrscheinlichkeit im Bereich $M - U_z \leq W \leq M + u_z$. Die gesamte Messunsicherheit ergibt sich zu $u = u_s + u_z$.

Die „Fehlergrenze“ g sagt aus, wie groß der Fehler dem Betrag nach höchstens werden darf. Dabei sollte der wahre Wert innerhalb des Intervalls $[x_a - g \leq W \leq x_a + g]$ liegen.

Immer wieder werden die beiden Begriffe „Auflösung“ und „Genauigkeit“ verwechselt. Oft werden Messgeräte oder Sensoren mit einer hohen Auflösung beworben, aber eine hohe Auflösung hilft nichts bei mangelhafter Genauigkeit. Es gilt:

Auflösung ist – vereinfachend – der minimale Unterschied zweier benachbarter Messwerte. Dabei spielt der absolute Betrag keine Rolle. Es werden nur relative Unterschiede betrachtet. Ein 8-Bit-A/D-Wandler mit 2,5 V Referenzspannung kann beispielsweise die Eingangsspannung in 10-mV-Schritten auflösen. Mit einem 10-Bit-A/D-Wandler und 2,5 V Referenzspannung kann man die Eingangsspannung bereits auf etwa 1 mV auflösen.

Genauigkeit gibt an, wie weit das Messergebnis vom wahren Wert abweicht. Es werden dabei absolute Messwerte betrachtet. Eine sogenannte Funkuhr wird immer die absolut richtige Zeit anzeigen, wo hingegen eine normale Quarzuhr hat möglicherweise 10 Sekunden Abweichung.[1]

1.2 Filterung von Messwerten

Wie oben schon erwähnt, haben eingelesene Analogwerte eine durch zufällige Ereignisse verursachte Streuung. Diese ergibt sich aus dem Umfeld des Controllers (elektromagnetische Störungen etc.), den Sensoreigenschaften (z.B. thermisches Rauschen) und der zu messenden physikalischen Größe. Insbesondere bei Anwendungen aus dem Bereich der Regelungstechnik ist ein Jitter bei den Eingangsdaten problematisch, denn dies könnte zum Schwingen des Reglers führen. Häufig ist also eine Glättung der Messwertreihe von Interesse.

Die einfachste Methode haben Sie schon kennen gelernt: es wird mehrmals nacheinander eingelesen und aus diesen Werten der Durchschnitt gebildet. Es wird dabei jedoch kein zeitlicher Bezug der Messwerte hergestellt, also kein Bezug des aktuellen Werts zu den vorher erfassten Werten. Bei fortlaufenden Messungen mit festem zeitlichen Abstand hingegen kann die Gesamtheit der Werte als Zeitreihe betrachtet werden.

Unter *Glättung* versteht man, dass die zufälligen Schwankungen der Werte einer Zeitreihe möglichst eliminiert werden sollen. Hierzu lassen sich mathematische Verfahren einsetzen, die eine gewichtete Mittelung der Werte innerhalb eines bestimmten Bereichs der Zeitreihe benutzen. Diese Verfahren werden Filter genannt. Dabei werden - ganz allgemein gesagt - neue Werte y_i gewonnen, indem die Messwerte x_i mit Gewichtsfaktoren a_i multipliziert und im Intervall (Fenster) zwischen den Grenzen i_u und i_o summiert werden:

$$y_i = \sum_{i=i_u}^{i_o} a_i * x_i \tag{1.3}$$

Die Gewichte a_i des Filters werden in der Regel durch Auswertung einer geeigneten deterministischen Filterfunktion bestimmt. Ein einfacher linearer Filter ist beispielsweise der Rechteckfilter, der alle Werte im jeweiligen Intervall gleich stark gewichtet. Für die Wahl $i_u = -i_o$ und $a_i =$

[1] Eine Kaputte Uhr zeigt wenigstens zweimal am Tag die richtige Zeit an.

$1/(i_o + i_u + 1)$ spricht man von einem einfachen **gleitenden Durchschnitt** der Messreihe oder auch einem „Moving-Average-Filter". Der symmetrische Dreiecksfilter wichtet die entsprechenden Werte innnerhalb eines Intervalls in linear abfallender Weise. Filter, die einen gleitenden Durchschnitt innerhalb eines Intervalls berechnen, bezeichnet man auch als Tiefpass-Filter, da das Signal auf kleinen zeitlichen Skalen (d. h. hohen Frequenzen) innerhalb der betrachteten Fensterbreite geglättet wird (Beispiel in Bild 1.1).

Für die Konstruktion eines Moving-Average-Filters bilden wir das arithmetische Mittel aus jeweils $2k + 1$ aufeinanderfolgenden Beobachtungswerten:

$$\overline{X}_t = \frac{1}{2k+1} * \sum_{i=-k}^{k} X_{t+i} \qquad mit \qquad t = k+1 \ldots N-k \tag{1.4}$$

Für einfache Anwendungen reicht es oft, immer nur den letzten Durchschnittswert für die Glättung heranzuziehen, es wird also der aktuelle Messwert gewichtet mit dem letzten Durchschnittswert zum neuen Durchschnittswert verknüpft. Dann wird auch kein Array benötigt. Das folgende Beispiel wichtet den neuen Wert mit 70 Prozent und den bis dahin ermittelten Durchschnitt mit 30 Prozent:

```
   ...
average = 0.7 * data + 0.3 * old_average;
old_average = average;
   ...
```

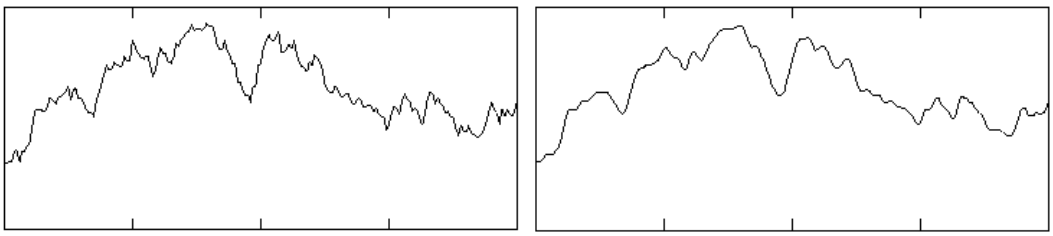

Bild 1.1: Grafische Darstellung einer Messwertreihe, links original, rechts mit gleitendem Durchschnitt

Es besteht ein einfacher Zusammenhang zwischen zwei aufeinanderfolgenden Werten, der sich für die Berechnung dieser Durchschnitte ausnutzen lässt: Bei Verschiebung des Zeitindex t ändert sich immer nur der erste und letzte Wert, über den die Mittelung erfolgt. Daraus ergibt sich:

$$\overline{X}_{t+1} = \overline{X}_t + \frac{1}{4k} * (-X_{t+1-k} + X_{t+1+k} - X_{t-k} + X_{t+k}) \tag{1.5}$$

Ein Nachteil der Moving-Average-Methode ist, dass am Anfang und am Ende Werte verloren gehen. Die Wahl des Mittelungsfensters k bestimmt, wie viele Werte in die Berechnung der gleitenden Durchschnitte eingehen. Dabei darf man k nicht zu groß wählen, weil sonst zu stark geglättet wird (insbesondere können hierdurch periodische Schwankungen weggemittelt werden). Bei Kurvenverläufen mit schwacher Krümmung und starker zufälliger Komponente verwendet man ein großes k und bei starker Krümmung und schwacher zufälliger Komponente ein niedriges k. Auf jeden Fall empfiehlt es sich, verschiedene Werte für k zu testen und denjenigen zu wählen, für den das Ergebnis am ehesten die gewünschten Eigenschaften besitzt.

Eine komplexere Möglichkeit der Glättung ist es, ein Polynom zu finden, das die Werte der Reihe möglichst gut annähert (polynomial regression), bei dem also die Summe der Fehlerquadrate (Abstand eine Punktes der Kurve vom entsprechenden Messwert) möglichst klein ist. Darüber hinaus gibt es weitere numerische Verfahren, beispielsweise die stückweise Interpolation der Werte durch kubische Splinefunktionen. Hierbei wird eine möglichst „glatte" Kurve durch die Werte gelegt. Normalerweise reicht aber der gleitende Durchschnitt, um „Ruhe" in die Kurve zu bringen. Will man dagegen Werteänderungen bei einem relativ glatten Kurvenverlauf herausarbeiten, bildet man die (gewichtete) Differenz benachbarter Werte.

1.3 Statistische Verarbeitung von Messwerten

Bei der Statistik handelt es sich um Verfahren, nach denen empirische Zahlen gewonnen, dargestellt, verarbeitet, analysiert und für Schlussfolgerungen, Prognosen und Entscheidungen verwendet wer-

den. Mit Statistik lassen sich keine Beweise führen, nur Hypothesen bekräftigen. Mit der mathematischen Statistik analysiert man Massenerscheinungen. Dabei zeigt sich oft, dass die Massenerscheinungen gewisse Gesetzmäßigkeiten aufweisen, die sich für Einzelerscheinungen nicht formulieren lassen, da sie dort nur als zufällige Unregelmäßigkeit auftreten. Werden z. B. 100 Werkstücke einzeln vermessen, so streuen die einzelnen Werte zufällig und sind somit nicht vorhersagbar. Die mittleren Maße und die Streuung der Werte jedoch sind auch nach dem Auszählen einer zweiten Stichprobe nahezu identisch. Diese charakteristischen Werte erlauben somit eine Aussage über die Grundgesamtheit aller Werkstücke. In der Wahrscheinlichkeitsrechnung wird der Mittelwert als Erwartungswert interpretiert.

Die Statistik kennt unterschiedliche Formen von Skalen:

Nominalskala: Die Ausprägungen sind Namen oder Kategorien, die aus technischen Gründen eine Nummer zugeordnet bekommen. Rechenoperationen lassen sich nicht sinnvoll durchführen. Beispiele: Geschlecht (0: männlich, 1: weiblich), Farben (rot: 1, blau: 2, gelb: 3, ...).

Ordinalskala: Ausprägungen können geordnet werden, aber die Abstände sind nicht interpretierbar, z. B. Noten (1 ist besser als 2).

Intervallskala: Hier können zwar die Abstände interpretiert werden, aber es gibt keinen interpretierbaren Nullpunkt. Beispiel: Temperaturmessung in Celsius-Graden. Man kann zwar sagen, es sei 10 Grad wärmer als vorige Woche, aber man kann nicht sagen, 20 Grad seien doppelt so warm wie 10 Grad.

Verhältnisskala: Das Verhältnis zweier Ausprägungen ist interpretierbar. Beispiel: Ein Einkommen von 4000 Euronen ist doppelt so hoch wie ein Einkommen von 2000 EUronen.

Intervall- und verhältnisskalierte Merkmale werden auch häufig als „metrisch“ bezeichnet.

1.3.1 Tabellarische und graphische Darstellung

Üblicherweise werden die n Beobachtungsereignisse (= Messwerte) der Reihe nach aufgelistet (Urliste). Aus dieser Stichprobe vom Umfang n kann man Schlüsse auf die zugehörige Grundgesamtheit N ziehen. Wären gleichzeitig mehrere Merkmale gemessen worden, z. B. Masse und Größe, so hätte man eine Stichprobe erhalten, die aus n Zahlenpaaren besteht.

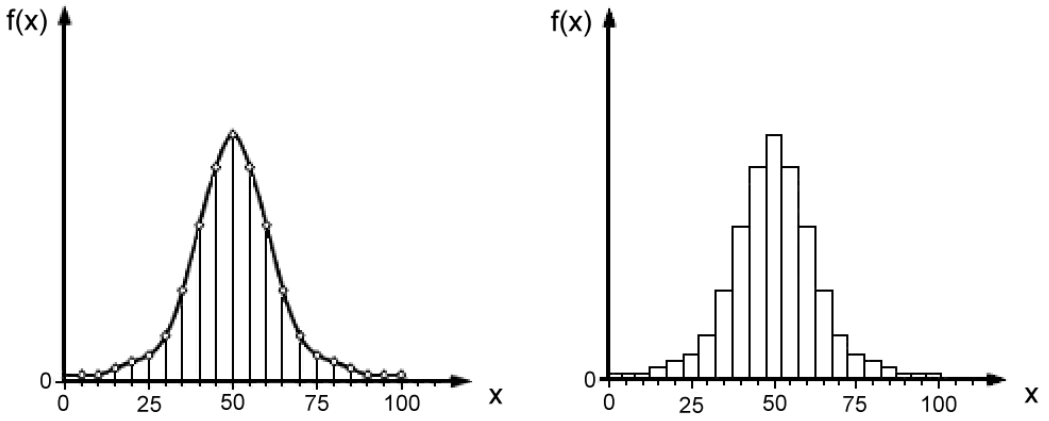

Bild 1.2: Häufigkeitsverteilung einer Gaußschen Normalverteilung, links: Stabdiagramm, Häufigkeitspolygon, rechts: Balkendiagramm

Bei kleinen Stichproben hilft es schon, wenn man die Werte der Größe nach ordnet, um einen Überblick zu bekommen. Besser ist es, zahlenmäßig gleiche Werte zusammenzufassen und sie graphisch darzustellen. Dabei wird die Anzahl a_i über dem Meßwert x_i aufgetragen. Die Anzahl a_i nennt man die absolute Häufigkeit des betreffenden Wertes in der Stichprobe. Dividiert man die absolute Häufigkeit durch den Stichprobenumfang n, so erhält man die relative Häufigkeit. Die relative Häufigkeit ist somit immer eine Zahl zwischen 0 und 1. Die Auftragung kann als Punkt-, Stab- oder Balkendiagramm erfolgen (Bild 1.2). Eine direkte Verbindung der Punkte untereinander ergibt ein Häufigkeitspolygon. Solche Graphiken stellen Häufigkeitsverteilungen oder Histogramme der Stichprobe dar.

Derartige statistische Methoden lassen sich auch zur Darstellung und Untersuchung von Messwerten verwenden.

Die folgende Funktion zeigt, wie einfach dies zu bewerkstelligen ist – sogar ohne Verwendung eines Grafikpakets. Sie druckt ein Histogramm für bis zu 25 Häufigkeiten, die im Array h gespeichert sind (`int werte[25]`). Dabei werden die Balken des Diagramms auf maximal 20 Zeilen Höhe normiert. Der erste Parameter der Funktion übergibt eine Überschrift, der zweite enthält das oben erwähnte Werte-Array und der dritte gibt an, wie viele Balken auszugeben sind.

```
void make_histo(char *headline, int werte[],  int n)
  {
  int h[25];
  int i, j, max = 0;

  for (i = 0; i <= n; i++)          /* Maximum finden */
    if (werte[i] > max) max = werte[i];
  for (i = 0; i <= n; i++)          /* Normieren */
    h[i] = werte[i]*20/max;
  printf("\n%s\n",headline);
  printf("  |");
  for (i = 0; i <= n; i++)          /* oberste Zeile */
    if (h[i] >= 20) printf("%2d ",h[i]);
    else printf("   ");
  printf("\n  ");
  for(j = 19; j >= 1; j--)          /* y-Achse */
    {
    printf("|");
    for (i = 0; i <= n; i++)        /* x- Werte */
      if (h[i]>=j) printf(" * ");
      else printf("   ");
    printf("\n  ");
    }
  for (i = 0; i <= n; i++)          /* x-Achse zeichnen */
    printf("%s","--+");
  printf("\n  ");
  for (i = 0; i <= n; i++)          /* x-Achse beschriften */
    printf("%3d",i);
  printf("\n");
  }
```

Für die statistische Auswertung eines Feldes mit Urdaten müssen zuerst die Häufigkeiten ermittelt werden. Dazu dient die folgende Funktion. Die Häufigkeiten werden in einem Feld „values“ gespeichert, dessen Komponenten den Datenwert und seine Häufigkeit speichern. Deshalb sind die Feldkomponenten als Verbund definiert:

```
struct Counter { double value; int count; };
```

Das Datenfeld muss aufsteigend sortiert vorliegen. Damit dies auch der Fall ist, wird die Bibliotheksfunktion `qsort` eingesetzt, die ihrerseits eine Vergleichsfunktion `dblvgl` benötigt:

```
int dblvgl(const double *v1, const double *v2)
  /* Vergleich zweier double-Werte (fuer Sortierung) */
  {
  if (*v1 > *v2) return(1);
  if (*v1 < *v2) return(-1);
  return (0);
  }

void frequencies(double data[], int n, struct Counter values[], int *toph)
  /* Zaehlt die Haeufigkeiten der Werte in data       */
  /* 'n' gibt die Anzahl der uebergebenen Werte an    */
  /* Rueckgabe: values: (Werte und zugeh. Haeufigk.), */
  /*    toph: letzter verwendeter Index von values    */
  /* Seiteneffekt: Sortieren des Datenfeldes          */
  {
  int i, k = 0;

  qsort(data, n, sizeof(data[0]), dblvgl);
  values[0].count = 1;
  values[0].value = data[0];
  for (i = 1; i < n; i++)
    {
    if (data[i] == data[i-1])
```

```
      values[k].count++;
    else
      {
      k++;
      values[k].count = 1;
      values[k].value = data[i];
      }
    }
  *toph = k;
  }
```

Um die nun ermittelten Häufigkeiten auszugeben, kann man statt des oben gezeigten senkrechten, alternativ ein waagrecht liegendes Histogramm drucken. Dies leistet die folgende Funktion:

```
void printfrequencies(struct Counter values[], int n)
  /* Ausgabe Histogramm der Haeufigkeiten in values  */
  /* 'n' gibt die Anzahl der uebergebenen Werte an   */
  /* struct Counter { double value; int count; };    */
  {
  int i, k, max = 0;
  /* maximum der Haeufigkeiten ermitteln */
  for (i = 0; i <= n; i++)
    if (values[i].count > max) max = values[i].count;
  for (i = 0; i <= n; i++)
    { /* Ausgabe */
    printf("%12lf: ", values[i].value);
    for (k = 0; k < values[i].count*50/max; k++)
      putchar('#');
    printf(" (%5d)\n",values[i].count);
    }
  }
```

Liegt ein diskretes oder stetiges Merkmal mit sehr vielen unterschiedlichen Ausprägungen vor, so ist es zweckmäßig, die Häufigkeiten nicht mehr jeder einzelnen Ausprägung zuzuordnen, sondern in (nicht überlappende, auch disjunkte) Klassen zusammenzufassen. Jede Klasse ist charakterisiert durch die untere und die obere Klassengrenze sowie die Klassenbreite. Die Klassenhäufigkeit ergibt sich aus der Anzahl der Elemente, deren Merkmalswert in der betreffenden Klasse liegen. Ein typisches Beispiel ist die Altersangabe in Fragebögen, hier wird nicht das Alter an sich erfasst, sondern gruppiert: 0 – 20, 20 – 40, 40 – 60, über 60. Damit das Skalenniveau erhalten bleibt, sind folgende Regeln zu beachten:

- Kategorien von Nominalskalen können beliebig zusammengefasst werden, wenn die Zuordnung der Messwerte eindeutig bleibt. Die Klassenbildung sollte inhaltlich begründet sein.
- Ordinalskalen können durch Zusammenlegen benachbarter Ränge vergröbert werden. Je größer die Variationsbreite der Messwerte ist, desto größer können die Klassen sein.
- Bei der Vergröberung von Intervallskalen ist darauf zu achten, dass die Klassenbreiten (kb) der Klassen bzw. Intervalle gleich sind. Die neu gebildete Klasse wird durch die Klassenmitte bezeichnet. Sie ist das arithmetische Mittel der unteren und der oberen Intervallgrenze.
- Bei Skalen von physikalischen Messungen, die nur selten über den gesamten theoretisch möglichen Bereich Messwerte liefern, kann es sinnvoll sein, am oberen und unteren Rand der Skala offene Messwertklassen zu bilden, in die alle Messwert bis bzw. ab einem bestimmten Messwert fallen. Bei Einbeziehung der offenen Klassen ist jedoch kein Intervallskalenniveau mehr gegeben.

1.3.2 Mittelwert, Varianz, Standardabweichung, Standardfehler

Neben der Häufigkeitsfunktion kann man eine Grundgesamtheit oder eine Stichprobe auch durch statistische Maßzahlen charakterisieren. Die beiden in der Praxis wichtigsten Maßzahlen sind der **Mittelwert**, der die durchschnittliche Größe der Stichprobe n kennzeichnet, und eine Angabe über die **Streuung** der Werte. Im Weiteren wird von der Annahme ausgegangen, dass die Messwerte eine Normalverteilung nach Gauß ergeben. Eine genügend große Stichprobe wird vorausgesetzt. Der arithmetische Mittelwert ist definiert als:

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n} * \sum_{i=1}^{n} x_i \tag{1.6}$$

Die folgende Funktion berechnet den Mittelwert eines `double`-Arrays. Als Parameter werden dieses Array und die Anzahl seiner relevanten Elemente übergeben (der Index läuft dann von 0 bis $n-1$):

```
double mittel(double data[], int n)
  /* Mittelwert des Arrays 'data' */
  /* 'n' gibt die Anzahl der uebergebenen Werte an */
  {
  int i;
  double sum;

  sum = 0.0;
  for (i = 0; i < n; i++)
    sum += data[i];
  return(sum/(double)n);
  }
```

Der Mittelwert allein reicht jedoch nicht aus, um z. B. eine Stichprobe zu beschreiben, wie folgendes Beispiel zeigt:

Stichprobe 1: `1.5, 3.0, 3.5, 4.0 -> = 3`
Stichprobe 2: `2.8, 2.9, 3.1, 3.2 -> = 3`

Beide Stichproben haben zwar den Mittelwert $\overline{x}$, unterscheiden sich aber voneinander, denn die Werte der ersten Stichprobe liegen viel weiter auseinander (und auch weiter vom Mittelwert entfernt) als die Werte der zweiten Stichprobe. Um diesen Unterschied zu erfassen, braucht man eine weitere Maßzahl, welche die Abweichung der Stichprobenwerte vom Mittelwert misst. Man bildet die Quadrate der Einzelabweichungen (kleinste Gaußsche Fehlerquadrate, engl. least square) und summiert diese auf. So erhält man die **Varianz** oder **Streuung** (engl. variance). Sie berechnet sich zu:

$$s^2 = \frac{1}{n-1} * \sum_{i=1}^{n} (x_i - \overline{x})^2 \tag{1.7}$$

Die positive Quadratwurzel der Varianz heißt **Standardabweichung** (engl. standard deviation). Sie gehört neben der Varianz zu den gebräuchlichsten Maßen zur Kennzeichnung der Variabilität einer Verteilung.

$$s = \sqrt{s^2} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}} \tag{1.8}$$

Für die obigen Beispiele ergeben sich somit für die erste Stichprobe $\overline{x}$ = 3, s^2 = 1.17 und s = 1,08 sowie für die zweite Stichprobe $\overline{x}$ = 3, s^2 = 0,03 und s = 0,18. Die Streuung der zweiten Stichprobe ist also wesentlich kleiner. Durch Angabe von Mittelwert und Varianz bzw. Standardabweichung sind Stichproben meist ausreichend beschrieben.

Bei einer symmetrischen Normalverteilung der Daten liegen zwei Drittel aller Fälle in einem Bereich $\overline{x} - s \leq x \leq \overline{x} + s$, was etwa 68 Prozent aller Fälle entspricht. In einer Entfernung von $\pm 2s$ liegen etwa 95 Prozent aller Fälle. Dieser Zusammenhang ist auch ein Grund dafür, dass Ausreißer in einer Verteilung, also Werte, die mehr als zwei Standardabweichungen vom Mittelwert entfernt liegen, oft durch diese Grenzwerte ersetzt werden.

Die Berechnung der Standardabweichung (bzw. Varianz) nach der obigen Formel ist jedoch numerisch ungünstig. Durch die Differenzbildung $x_i - \overline{x}$ entstehen sehr kleine Differenzen, die dann auch noch quadriert werden müssen. Durch Rundungsfehler kommt es zu Genauigkeitsverlusten. Deshalb gibt es für die Praxis günstigere Berechnungsformeln. Bei ihnen wird die Differenzbildung vermieden. Für die Standardabweichung einer Stichprobe ergibt sich somit:

$$s = \sqrt{\frac{\sum_{i=1}^{n} x_i^2 - n * \overline{x}^2}{n-1}} \tag{1.9}$$

Die folgende Funktion `varianz` hat die gleichen Parameter wie die Mittelwert-Funktion. Es werden in der Schleife gleichzeitig Summe und Quadratsumme berechnet und anschließend die Varianz.

```
double varianz(double data[], int n)
  /* Varianz des Arrays 'data' */
  /* 'n' gibt die Anzahl der uebergebenen Werte an */
  {
  int i;
  double sum, sumsq;

  sum = 0.0;
  sumsq = 0.0;
  for (i = 0; i < n; i++)
    {
    sum += data[i];               /* Summe */
    sumsq += (data[i] * data[i]); /* Summer der Quadrate */
    }
  return((sumsq - sum*sum/(double)n)/(double)(n - 1));
  }
```

Bei der Bearbeitung von Stichproben (= Messwerten) will man oft wissen, mit welcher Wahrscheinlichkeit von den bei einer Stichprobe gefundenen Größen auf die Grundgesamtheit geschlossen werden kann. Die für eine Stichprobe ermittelten Werte (Mittelwert, Varianz, Standardabweichung) sind also nur Schätzwerte für die Grundgesamtheit. Man möchte z. B. wissen, wie weit der Stichprobenmittelwert vom Mittelwert der Grundgesamtheit abweicht. Diese Abweichung bezeichnet man als **Standardfehler** (= Fehler des Mittelwertes = Standardabweichung des Mittelwertes; engl. standard error of the mean). Wenn keine extremen Abweichungen der Stichprobenwerte x_i von der Normalverteilung um den Stichprobenmittelwert vorliegen, darf man annehmen, dass sich auch die Mittelwerte annähernd gleich großer Stichproben gleichmäßig um den Mittelwert der Grundgesamtheit verteilen. Die unbekannte wirkliche Abweichung kann durch den Standardfehler $s_{\overline{x}}$ abgeschätzt werden. Er berechnet sich aus der Standardabweichung s zu:

$$s_{\overline{x}} = \frac{s}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n * (n-1)}} \tag{1.10}$$

Das zusätzliche n in der Formel für den Standardfehler (im Gegensatz zu Varianz und Standardabweichung) liefert eine Angabe über die Größe der Stichprobe. Je größer eine Stichprobe ist, desto genauer wird die Schätzung für die Grundgesamtheit. Der Standardfehler verkleinert sich dabei (n steht im Nenner), geht somit gegen den Mittelwert der Grundgesamtheit.

Die C-Funktion dazu ist trivial – sie berechnet lediglich die Quadratwurzel der oben gezeigten Varianz-Funktion:

```
double streuung(double data[], int n)
  /* Streuung des Arrays 'data' */
  /* 'n' gibt die Anzahl der uebergebenen Werte an */
  {
  return(sqrt(varianz(data,n)));
  }
```

Der Standardfehler wird oft zusammen mit dem Stichprobenmittelwert zur Charakterisierung einer Stichprobe bezüglich der Grundgesamtheit angegeben, z. B. als $\overline{x} \pm s_{\overline{x}}$, etwa 25 cm $\pm$ 0,1 cm).

1.3.3 Minimum, Maximum, Median, Modalwert

Minimum ist der kleinste, **Maximum** der größte im Array vorkommende Wert. Der Zentralwert oder **Medianwert** stellt ebenfalls einen charakteristischen Wert einer Häufigkeitsverteilung dar. Er teilt die Häufigkeitsverteilung flächengleich auf, so dass sich links und rechts vom Zentralwert genau gleich viele Ereignisse befinden. Der häufigste Wert oder **Modalwert** stellt, wie sein Name schon sagt, den Wert mit der größten Häufigkeit dar. Er ist also der Gipfel in einer Häufigkeitsverteilung. In einer Normalverteilung sind infolge der Symmetrie der Verteilung arithmetischer Mittelwert, Medianwert und Modalwert identisch. Er kann mit Hilfe der weiter oben vorgestellten Funktion `frequencies` ermittelt werden.

Die Funktion zum Berechnen der anderen Werte muss das Datenfeld sortieren (speziell für den Median). Dazu wird wieder die Bibliotheksfunktion `qsort` verwendet. Die statistischen Werte gibt die Funktion als Referenzparameter zurück:

```
int dblvgl(const double *v1, const double *v2)
  /* Vergleich zweier double-Werte (fuer Sortierung) */
  {
  if (*v1 > *v2) return(1);
  if (*v1 < *v2) return(-1);
  return (0);
  }

void minimax(double data[], int n,
             double *min,
             double *max,
             double *median)
  /* Minimum, Maximum und Median des Arrays 'data'  */
  /* 'n' gibt die Anzahl der uebergebenen Werte an  */
  /* Rueckgabe: min, max, median                    */
  /* (data ist anschliessend sortiert)              */
  {
  qsort(data, n, sizeof(data[0]), dblvgl);
  *min = data[0];
  *max = data[n-1];
  if ((n % 2) == 0) /* n gerade */
    *median = (data[(n-1)/2] + data[(n-1)/2+1])/2;
  else
    *median = data[(n-1)/2+1];
  }
```

Das folgende Hauptprogramm zeigt, wie man die oben aufgeführten Funktionen aufruft. Der Einfachheit halber wurde das Datenfeld statisch vorbelegt, bei einer realen Anwendung wird man die Daten entweder im selben Programm vom A/D-Wandler erhalten oder aus einer Datei einlesen.

```
int main(void)
  {
  /* Statistik-Demoprogramm */
  double Data[MAXDATA] = {1.5, 2.0, 1.5, 3.0, 2.5, 1.5, 2.0, 2.5, 3.5, 3.0};
  struct Counter Hauf[MAXDATA];
  double min, max, median;
  int toph;

  printf("Mittel: %lf Varianz: %lf Streuung: %lf\n",
          mittel(Data,10), varianz(Data,10), streuung(Data,10));

  minimax(Data,10, &min, &max, &median);
  printf("Minimum: %lf Maximum: %lf Median: %lf Spanne: %lf\n",
          min, max, median,max-min);

  frequencies(Data,10,Hauf,&toph);
  printfrequencies(Hauf,toph);
  return(0);
  }
```

Die Ausgabe des Programms sieht dann beispielsweise folgendermaßen aus:

```
Mittel: 2.300000 Varianz: 0.511111 Streuung: 0.714920
Minimum: 1.500000 Maximum: 3.500000 Median: 2.250000 Spanne: 2.000000

    1.500000: ################################################## (    3)
    2.000000: ################################## (    2)
    2.500000: ################################## (    2)
    3.000000: ################################## (    2)
    3.500000: ################ (    1)
```

1.3.4 Mittelwert und Varianz ohne Array

Gerade bei Mikrocontrollern ist das verfügbare RAM relativ schmalbrüstig. Bei manchen 8-Bit-CPUs gibt es da gerade mal 8 KByte. Bei vielen Anwendungen der Messwerterfassung reicht das nicht aus, um nach Standardverfahren die Daten erst in einem Array zu sammeln und dieses dann auszuwerten. Auch kann diese Form der Datenverarbeitung den Datenfluss empfindlich stören, da die Berechnung von Mittelwert und ggf. Standardabweichung eine gewisse Zeit beansprucht.

Zur Illustration soll ein Beispiel dienen: Über eine entsprechende Schnittstelle werden Sounddaten eingelesen und sollen ausgewertet werden. Dabei geht es nicht darum, die Geräusche 1:1 aufzuzeichnen, sondern es soll nur festgestellt werden, ob der Lärm einen gewissen Pegel überschreitet. Dabei sind auch einzelne Signalmaxima nicht von Belang, die möglicherweise im Millisekundenbereich

auftreten, sondern es geht um den mittleren Pegel. Daher wird das Sound-Signal über eine Sekunde gemittelt und nur dieser Mittelwert betrachtet. Bei minimaler Soundqualität sind das immer noch 8000 Samples/Sekunde, bei HiFi-Qualität wären es schon 44000 Samples/Sekunde. Wenn man die Werte als 16-Bit-Integer speichert, belegt eine Sekunde dann schon 88000 Bytes – da reicht das RAM eines Controllers nicht mehr aus.

Um das Ganze ohne Array zu realisieren („stream processing“), wird die Standard-Formel für den Mittelwert umgeformt. Angenommen der n-te Datenwert sei a_n. Der bis dahin ermittelte Mittelwert sei m_n. Damit gilt:

$$m_n = \frac{1}{n} \sum_{i=1}^{n} a_i \tag{1.11}$$

Wählt man als Amnfangswert $m_0 = 0$, so kann man die obige Gleichung umformen:

$$m_n = \frac{(n-1)m_{n-1} + a_n}{n} \tag{1.12}$$

Um Rundungsfehler zu minimieren (es kann sein, dass bei großem n der Wert von $(n-1)m_{n-1}$ sehr viel größer als a_n ist), wird nochmals umgeformt:

$$m_n = m_{n-1} + \frac{a_n - m_{n-1}}{n} \tag{1.13}$$

Damit werden nur noch zwei Werte benötigt, m_n und m_{n-1}. In einer Programmschleife wird zunächst der neue Wert m_n berechnet und dann $m_{n-1} = m_n$ gesetzt.

Die Mittelwertberechnung ist damit serialisiert. Falls noch die Varianz benötigt wird, hilft ein Blick in die Literatur – in diesem Fall ein Standardwerk aus den 1970er Jahren: „The Art of Computer Programming, Volume 2: Seminumerical Algorithms“ von Donald Knuth. Dort werden die folgenden Formeln auch erklärt und hergeleitet. Der Mittelwert wird berechnet wie oben gezeigt:

$$m_n = m_{n-1} + \frac{a_n - m_{n-1}}{n} \tag{1.14}$$

Für die Varianz läuft bei der seriellen Berechnung die Variable s_n mit:

$$s_n = s_{n-1} + (a_n - m_{n-1})(a_n - m_n) \tag{1.15}$$

Die Varianz erhält man dann, indem der Stream-Wert s_n durch $(k-1)$ dividiert wird. Die Wurzel daraus ergibt dann die Standardabweichung.

Das folgende Listing zeigt eine Muster-Implementierung in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

/*
 * compile with: gcc -Wall -o stat stat.c -lm
 */

/*
 * Daten-Stucture fuer die globalen Werte. Eine entsprechende
 * Variable wird an die Funktionen uebergeben (Achtung: bei
 * Clear() und Store() als Pointer).
 */
struct Stat_Rec
  {
  int stat_n;        /* Wertezaehler */
  double stat_oldM;  /* Mittelwert alt und neu */
  double stat_newM;
  double stat_oldS;  /* Varianz-Basis alt und neu */
  double stat_newS;
  };

/*
 * Initiieren einer neuen Messung, Setzen aller Variablen auf 0
 */
```

```
void Clear(struct Stat_Rec * SR)
  {
  SR->stat_n = 0;
  SR->stat_oldM = 0.0;
  SR->stat_newM = 0.0;
  SR->stat_oldS = 0.0;
  SR->stat_newS = 0.0;
  }

/*
 * Hinzufuegen eines neuen Messwert, serialisierten Berechnung
 * von Mittelwert und Varianz-Basis, Hochzaehlen des Zaehlers.
 * Implementierung des Algorithmus von D. Knuth.
 */
void Store(double x, struct Stat_Rec * SR)
  {
  SR->stat_n++;
  if (SR->stat_n == 1)
    {
    SR->stat_oldM = x;
    SR->stat_newM = x;
    SR->stat_oldS = 0.0;
    SR->stat_newS = 0.0;
    }
  else
    {
    SR->stat_newM = SR->stat_oldM + (x - SR->stat_oldM)/SR->stat_n;
    SR->stat_newS = SR->stat_oldS + (x - SR->stat_oldM)*(x - SR->stat_newM);
    SR->stat_oldM = SR->stat_newM;
    SR->stat_oldS = SR->stat_newS;
    }
  }

/*
 * Abfrage des aktuellen Zaehlerstandes
 */
int NumDataValues(struct Stat_Rec SR)
  {
  return SR.stat_n;
  }

/*
 * Abfrage des aktuellen Mittelwerts
 */
double Mean(struct Stat_Rec SR)
  {
  return ( (SR.stat_n > 0) ? SR.stat_newM : 0.0 );
  }

/*
 * Abfrage der aktuellen Varianz
 */
double Variance(struct Stat_Rec SR)
  {
  return ( (SR.stat_n > 1) ? SR.stat_newS/(SR.stat_n - 1) : 0.0 );
  }

/*
 * Abfrage der aktuellen Standardabweichung
 */
double StdDeviation(struct Stat_Rec SR)
  {
  return sqrt( Variance(SR) );
  }


/*
 * Testprogramm zur Illustration der Anwendung
 */
int main(void)
  {
  struct Stat_Rec Stat;

  double mean, variance, stddev;
```

```
Clear(&Stat);

Store(18.0, &Stat);
Store(16.0, &Stat);
Store(18.0, &Stat);
Store(18.0, &Stat);
Store(20.0, &Stat);

mean = Mean(Stat);
variance = Variance(Stat);
stddev = StdDeviation(Stat);
printf ("Mean: %5.2lf, Var.: %5.2lf, StdDev.: %5.2lf\n",
       mean, variance, stddev);
return 0;
}
```

Ein Probelauf liefert erwartungsgemäß `Mean: 18.00, Var.: 2.00, StdDev.: 1.41`.

1.3.5 Lineare Regression

Bei der Erfassung von Meßgrößen besteht oft der Wunsch, Werte zwischen den einzelnen Stützpunkten zu ermitteln, beispielsweise um eine Kurve der Werte darzustellen. Hier helfen Interpolation und Regression.

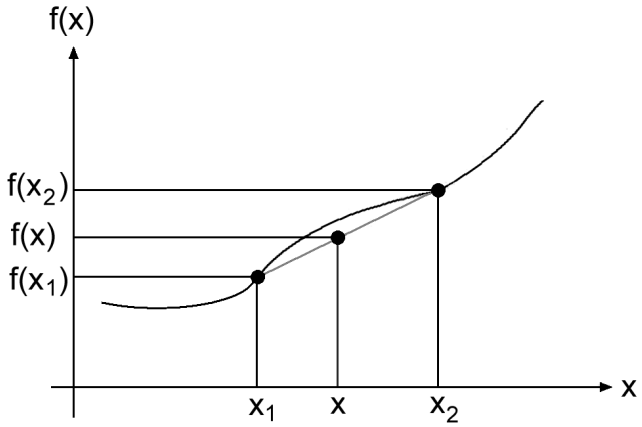

Bild 1.3: Interpolation: Näherung des Funktionswertes durch eine Gerade zwischen zwei Stützstellen

Bei der **linearen Interpolation** wird die Näherung eines Funktionswerts **f(x)** mit $x_1 < x < x_2$ aus den bekannten Funktionswerten $f(x_1)$ und $f(x_2)$ gewonnen. Bild 1.3 veranschaulicht die Ermittlung eines Zwischenwertes.

Sind von einer Funktion nur die Werte an den Stützstellen $x_1, x_2, \ldots, x_n$ bekannt, so lässt sich durch Interpolation eine stückweise lineare Funktion definieren. Für die Interpolation eines Wertes x zwischen zwei Stützstellen x_1 und x_2 gilt die Gleichung:

$$f(x) = \frac{(x_2 - x)f(x_1) + (x - x_1)f(x_2)}{x_2 - x_1} \tag{1.16}$$

Bei der **linearen Regression** wird eine Gerade $y = ax + b$ gesucht, die in bestmöglicher Näherung durch die n Datenpunkte $(x_1, f(x_1))$, $(x_2, f(x_2))$, …,$(x_n, f(x_n))$ gelegt wird. Die bestmögliche Anpassung ist gegeben, wenn die Summe der Abweichungsquadrate, $\sum_{i=1}^{n}(ax_i + b - f(x_i))^2$, minimal wird. Die Werte von a und b errechnen sich aus den folgenden Gleichungen:

$$a = \frac{n * \sum_{i=1}^{n}(x_i * f(x_i)) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} f(x_i))}{n * (\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x_i)^2} \tag{1.17}$$

$$b = \frac{\sum_{i=1}^{n} f(x_i) - a * (\sum_{i=1}^{n} x_i)}{n} \tag{1.18}$$

Im folgenden Programm wird a in die Formel für b eingesetzt – daher sieht es etwas anders aus. Zudem werden nicht nur die Parameter a und b errechnet, sondern auch zwei statistische Maßzahlen, die es erlauben, die „Güte“ der Regressionsgeraden zu beurteilen. Es sind dies die mittlere quadratische Abweichung der Geraden von allen Stützpunkten und die Standardabweichung für a und b.

Zum Testen des Programms wurden die Funktionswerte der Gleichung $f(x) = 2a + 2$ leicht verändert.

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

int linreg (double x[], double y[], int anz,
            double *a, double *b,
            double *r, double *sa, double *sb)
/* Lineare Regression fuer 'anz' Stuetzpunkte (x[i], y[i]):
*   Rueckgabeparameter (alles Referentparameter):
*   a, b: Koeffizienten der Regressionsgerade y = ax + b
*   r: mittlere quadratische Abweichung
*   sa, sb: Standardabweichungen fuer a und b
*   Die Funktion liefert im Erfolgsfall 0, wenn die
*   Koeffizienten nicht berechnet werden koennen, 1.
*   Aufruf: linreg(x,y,anzahl,&a,&b,&r,&sa,&sb);
*/
  {
  double summexy = 0.0, summex = 0.0;
  double summey = 0.0, summex2 = 0.0;
  double sum = 0.0;
  double divisor = 0.0;
  double n = (double) anz;
  int i;

  for(i = 0; i < anz; i++)
    {
    summex += x[i];
    summey += y[i];
    summexy += (x[i] * y[i]);
    summex2 += (x[i] * x[i]);
    }
  divisor = (n*summex2 - summex*summex);

  if (divisor < 1.0E-30) return (1); /* Division durch 0! */

  /* a und b fuer y = ax + b berechnen */
  *a = (n*summexy - summex*summey)/divisor;
  *b = (summex2*summey - summex*summexy)/divisor;

  /* mittlere quadratische Abweichung */
  for(i = 0; i < anz; i++)
    sum += pow(*a*x[i] + *b - y[i],2);
  *r = sqrt(sum/(n - 2.0));

  /* Standardabweichung von a und b */
  *sa = *r*sqrt(n/divisor);
  *sb = *r*sqrt(summex2/divisor);

  return (0);
  }

int main(void)
  {
  double x[] = {1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0};
  double y[] = {4.1,6.1,7.9,9.9,12.1,13.8,16.0,18.2,19.9,22.1};
  int anz = 10;
  double a, b, r, sa, sb;

  if(linreg(x,y,anz,&a,&b,&r,&sa,&sb) != 0)
    printf("Nicht berechenbar!\n");
  else
    {
    printf("Regressionsgerade: y = %lf * x + %lf\n",a,b);
    printf("Mitt. quad. Abweichung: %lf\n",r);
    printf("Standardabweichung a:   %lf\n",sa);
    printf("Standardabweichung b:   %lf\n",sb);
```

```
    }

return 0;
}
```

1.4 Ausreisser-Test

In der Statistik spricht man von einem Ausreißer, wenn ein Messwert oder Befund nicht in eine erwartete Messreihe passt, physikalisch unmöglich ist oder allgemein nicht den Erwartungen entspricht. Der Bereich für die Erwartungswerte wird meist als Streuungsbereich des Wertes definiert, in dem die Mehrzahl der Messwerte liegt, z. B. der Quarilabstand Q75 – Q25. Werte, die weiter als das 1,5-fache des Quartilabstandes außerhalb dieses Intervalls liegen, werden (meist willkürlich) als „Ausreißer" bezeichnet. Die robuste Statistik beschäftigt sich mit der Ausreißerproblematik. Auch beim Data-Mining beschäftigt man sich mit dem Erkennen von Ausreißern. Entscheidend ist es in jedem Fall, zu überprüfen, ob es sich bei den Ausreißern tatsächlich um ein verlässliches und echtes Ergebnis handelt, oder ob ein Messfehler vorliegt.

Der Vergleich des Medians mit dem arithmetischen Mittel kann als Ausreißeranalyse verwendet werden. Weicht der Median auffällig vom arithmetischen Mittel ab, sollten die Daten auf Ausreißer oder stark schiefe Verteilungen hin überprüft werden. Für das richtige Verständnis der Daten ist es wichtig, dass man weiss, wie man Ausreißer berechnet und richtig bewertet. Dadurch lassen sich ggf. präzisere Schlussfolgerungen aus statistischen Daten ziehen.

Vor der Entscheidung, ob ein ungewöhnlicher Wert in einer gegebenen Datenreihe vernachlässigt werden kann, muss der mögliche Ausreißer identifiziert werden. Ausreißer sind scheinbar recht einfach in Tabellen oder in Grafiken zu erkennen. Sind die Daten in einer Grafik dargestellt, liegen Ausreißer weit entfernt von den anderen Werten. Nicht immer ist ein Ausreißer ein zu vernachlässigender Wert. Untersucht man beispielsweise das Wärmebild eines elektrischen Verteilerschranks und stellt fest, dass alle Messpunkte um die 25 Grad liegen, aber ein Messpunkt 70 Grad aufweist, so darf man in diesem Fall den Ausreißer nicht ignorieren - er ist vielmehr der wichtigste Messpunkt, denn er weist auf eine defekte Verbindung hin, die sich übermäßig erwärmt hat. Erst wenn die Frage geklärt ist, ob es sich bei den möglichen Ausreißern um einen sinnvollen Messwert handelt oder nicht, kann weitergemacht werden.

Um Ausreißer in einem Datensatz zu ermitteln, ist den Median ein robusterer Mittelwert als das arithmetische Mittel. Der Median markiert die Mitte der aufsteigend nach der Größe geordneten Datenwerte. Als Beispiel soll der folgende Datensatz dienen (es handelt sich um die Werte eines Ultraschall-Entfernungssensors):

```
59.0, 60.7, 60.5, 60.5, 633.1, 60.1, 60.8, 59.8, 59.7, 59.3, 60.5, 60.6, 60.5,
60.6, 59.2, 59.4, 59.2, 60.2, 58.6, 59.8, 59.9, 60.0, 60.2, 60.2, 61.3
```

Sortiert ergibt sich folgende Datenreihe:

```
58.6
59.0
59.2
59.2
59.3
59.4
59.7
59.8
59.8
59.9
60.0
60.1
60.2  ___ Median
60.2
60.2
60.5
60.5
60.5
60.5
60.6
60.6
60.7
60.8
```

```
61.3
633.1
```

Der Median ist derjenige Datenpunkt, der sich genau in der Mitte der Datenreihe befindet. Er teilt die Datenreihe in zwei Hälften. Falls die Datenreihe wie im Beispiel über eine ungerade Anzahl an Werten verfügt, ist es der Wert, der genauso viele Werte über sich wie unter sich hat. Hat die Datenreihe dagegen eine gerade Anzahl an Werten hat, wird der Durchschnitt der beiden mittleren Werte gebildet. Im Beispiel hat der Median den Wert 60.2. Bei der Berechnung von Ausreißern wird dem Median oft der Variablenname Q2 zugeordnet, da er zwischen dem unteren Quartil Q1 und dem oberen Quartil Q3 liegt.

Die beiden Qartile Q1 und Q2 werden folgendermaßen ermittelt: Das untere Quartil Q1 ist der Datenpunkt, unter dem sich 25% (ein Viertel) der Messwerte befinden. Das ist der halbe Bereich von Werten unter dem Meridian. Ergibt sich hier eine gerade Anzahl von Werten, wird wieder der Durchschnitt der beiden mittleren Werte um dies Stelle herum gebildet. Im Beispiel liegen 12 Werte unterhalb des Median. Es muss also der Durchschnitt aus den beiden Werten 59.4 und 59.7 gebildet werden, um das untere Quartil zu finden: Q1 = (59.4 + 59.7)/2 = 59.55 (siehe unten).

Auf die gleiche Weise wird das obere Quartil Q3 berechnet, nur diesmal in der oberen Hälfte der Datenwerte: Hier handelt es sich um die Werte 60.5 und 60.6: Q3 = (60.5 + 60.6)/2 = 60,55.

```
58.6
59.0
59.2
59.2
59.3
59.4  ___ Q1 = 59,55
59.7
59.8
59.8
59.9
60.0
60.1
60.2  ___ Median Q2 = 60.2
60.2
60.2
60.5
60.5
60.5
60.5  ___ Q3 = 60,55
60.6
60.6
60.7
60.8
61.3
633.1
```

Nachdem Q1 und Q3 bestimmt wurden, kann der Abstand zwischen diesen beiden Werten, der Interquartilsabstand, berechnet werden, indem Q1 von Q3 subtrahiert wird. Dieser Wert ist wichtig für die Festlegung der Grenzen für Ausreißer in der Datenreihe: I = Q3 - Q1 = 60.55 - 59.55 = 1.0.

Nun wird untersucht, ob sich innerhalb bestimmter zahlenmäßiger Grenzen Datenwerte befinden. Auch die Art der Ausreißer wird in „milde“ und „extreme“ Ausreißer klassifiziert. Dazu bildet man zwei Grenzen, genannt „innerer Zaun“ und „äußerer Zaun“. Ein Wert, der außerhalb des inneren Zauns liegt, ist ein „milder Ausreißer“. Ein Wert, der hingegen außerhalb des äußeren Zauns liegen, gilt als „extremer Ausreißer“.

- Um den inneren Zaun der Datenreihe zu finden, wird der Interquartilsabstand mit 1.5 multipliziert. Dann wird das Ergebnis zu Q3 addiert und von Q1 subtrahiert. Die beiden daraus entstehenden Werte sind die die Grenzen des inneren Zauns der Datenreihe. Im Beispiel:

  ```
  Q3 + 1.5*1.0 = 60.55 + 1.5 = 62.05
  Q1 - 1.5*1.0 = 59.55 - 1.5 = 58.05
  ```

- Der äußere Zaun wird auf dem gleichen Weg ermittelt, wie der inneren Zaun, nur dass hier der Interquartilsabstand mit 3.0 multipliziert wird. Um die untere und obere Grenze unseres äußeren Zauns zu finden, wird das Ergebnis dann wieder zu Q3 addiert und von Q1 subtrahiert. Im Beispiel:

  ```
  Q3 + 3.0*1.0 = 60.55 + 3.0 = 63.55
  Q1 - 3.0*1.0 = 59.55 - 3.0 = 56.55
  ```

Für die Datenreihe ergibt sich dann:

```
       ___ äußerer Zaun = 56.55
       ___ innerer Zaun = 58.05
  58.6
  59.0
  59.2
  59.2
  59.3
  59.4  ___ Q1 = 59,55
  59.7
  59.8
  59.8
  59.9
  60.0
  60.1
  60.2  ___ Median Q2 = 60.2
  60.2
  60.2
  60.5
  60.5
  60.5
  60.5  ___ Q3 = 60,55
  60.6
  60.6
  60.7
  60.8
  61.3  ___ innerer Zaun = 62.05
       ___ äußerer Zaun = 63.55
 633.1
```

Jeder Punkt außerhalb des äußeren Zauns wird als extremer Ausreißer betrachtet. Im Beispiel liegt die letzte gemessene Distanz (633.1) deutlich außerhalb des äußeren Zauns, ist also ein extremer Ausreißer.

Mit der oben beschriebenen Methode kann ermittelt werden, ob ein bestimmter Wert ein milder, ein extremer oder gar kein Ausreißer ist. Wenn ein Wert als Ausreißer identifiziert wurde, heißt das noch lange nicht, dass man ihn auch vernachlässigen darf. Der Grund, warum ein Ausreißer vom Rest der Werte in einer Reihe abweicht, ist das entscheidende Kriterium dafür, ob er vernachlässigt werden kann oder nicht. Ausreißer, die auf einen Fehler irgendeiner Art zurückzuführen sind (Fehler bei der Messung, des Versuchsaufbaus etc.), kann man meist weglassen. Ausreißer, die nicht als Fehler identifiziert werden können, darf man dagegen nicht weglassen.

Als weiteren Anhaltspunkt kann das arithmetische Mittel herangezogen werden. Wie eingangs erwähnt, kann der Vergleich des Medians mit dem arithmetischen Mittel als Ausreißeranalyse verwendet werden. Weicht der Median auffällig vom arithmetischen Mittel ab, sollten die Daten auf Ausreißer oder stark schiefe Verteilungen hin überprüft werden. Das arithmetische Mittel ergibt 80.6, was sich signifikant vom Median (60.2) unterscheidet. Läßt man den Ausreißer 633.1 weg, ergibt sich ein arithmetisches Mittel von 60.0, das nahe am Median liegt.

2

Grafik-Tools für die Messwert-Darstellung

2.1 Programmierung grafischer Darstellungen

2.1.1 Die GD-Bibliothek

Sie können mit relativ wenig Aufwand unter Zuhilfenahme des GD-Pakets von Thomas Boutell (*http://www.boutell.com/gd/*) Ihre Messwerte mit selbst programmierten Grafiken darstellen. Die Bibliothek ist nicht nur für C/C++ verfügbar, sondern auch für Programmiersprachen wie Perl oder PHP. Insbesondere letztere erlaubt die programmgesteuerte Generierung von Grafiken auf einer Webseite praktisch „on the fly". Nachteilig ist dabei, dass Sie aufbauend auf Grafik-Basisfunktionen wie „zeichne Punkt an der Bildschirmposition X,Y" oder „zeichne Linie von (X1,Y1) nach (X2,Y2)" alles selbst machen müssen: Skalierung mit Beschriftung und Skalenmarken, Linien ziehen, Boxen zeichnen und ggf. farbig füllen und vieles mehr.

Die GD-Bibliothek benötigt ihrerseits die Bibliotheken Libpng und Zlib. Sie müssen also zunächst die drei Bibliotheken installieren und – wenn Sie die GD-Library in Perl verwenden wollen – das GD-Modul von Perl. Insbesondere bei Perl müssen Sie sich nicht mit Grafikprimitiven herumschlagen, sondern können sich zahlreicher anderer Perl-Module bedienen, die das Zeichnen von zwei- oder dreidimensionalen Charts usw. ermöglichen. Beispielsweise nimmt uns das Chart-Modul von David Bonner fast die ganze Arbeit ab. Die unterschiedlichen Chart-Typen (Points, Lines, Bars, LinesPoint, StackedBars, Pie etc.) sind untereinander kompatibel, so dass man durch Ändern eines einzigen `use`-Kommandos die Art der Darstellung wechseln kann.

2.1.2 Beispiel für die Programmierung

Das folgende Perl-Programm ermöglicht es, schnell mal nebenbei Datenreihen zu visualisieren. Mit der Kommandozeile `chart.pl < data` wird das Programm ausgeführt. Die Datei *`data`* hat folgenden Aufbau, wobei einzelne Items innerhalb einer Zeile durch Strichpunkte getrennt werden (Leerzeichen rechts und links vom Strichpunkt stören nicht):

1. Zeile: Dateiname (ohne Endung) der Ausgabedatei für die Grafik
2. Zeile: Überschrift der Chart, danach der Typ der Grafik („L" für Linien, „B" für Balken oder „p" für Punkte), gegebenenfalls gefolgt von einem „g", falls Gitterlinien gewünscht werden. Werden diese Angaben weggelassen, erzeugt das Programm eine Liniengrafik ohne Gitterlinien.
3. Zeile: Überschrift der Y-Achse, bei Bedarf gefolgt von Minimum/Maximum der Y-Achsenskala (z. B. Temperatur °C;0;150)
4. Zeile: Überschrift der X-Achse
5. Zeile: X-Achsen-Tics, d. h. die Werteskala der X-Achse. Hier müssen genauso

viele Werte angegeben werden, wie Zahlenwerte in den folgenden Zeilen (z. B. 0; 300; 600; 900; 1200)

In den darauf folgenden Zeilen stehen die Datenwerte für eine Kurve. Der erste Wert dient als Legende, danach folgen die einzelnen Werte.

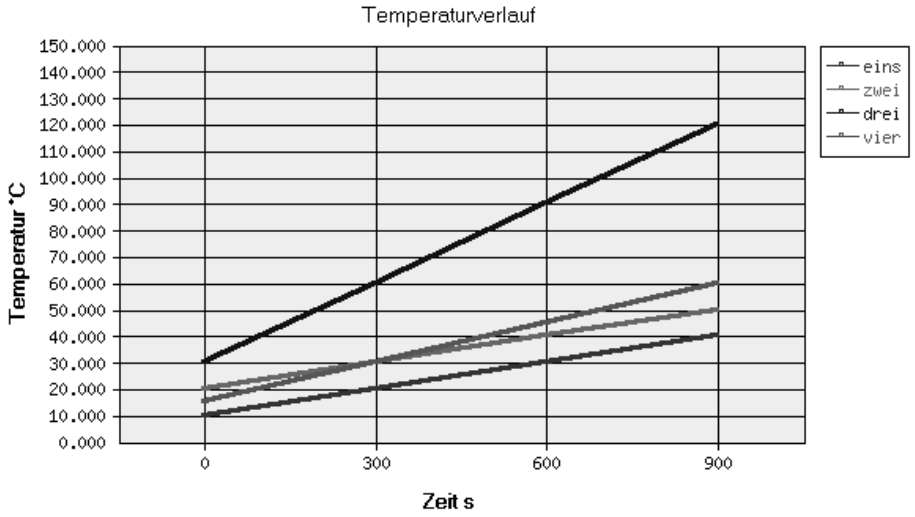

Bild 2.1: Darstellung von Messwerten mit dem Chart-Modul

Eine Eingabedatei für das Chart-Programm könnte beispielsweise aussehen, wie im folgenden Listing gezeigt. Aus den Daten wird die Grafik in Bild 2.1 erzeugt, wobei die Farben der Linien hier nur als Grauwerte zu erkennen sind.

```
Data
Temperaturverlauf;p;n
Temperatur °C;0;150
Zeit s
0;300;600;900
eins;10;20;30;40
zwei;20;30;40;50
drei;30;60;90;120
vier;15;30;45;60
```

Diese Datei lässt sich relativ leicht bei der Messung generieren und bietet gleichzeitig die Möglichkeit des Imports als CSV-Datei in andere Programme. Das Perl-Programm zum Verarbeiten dieser Eingabedaten ist nicht besonders umfangreich:

```
#!/usr/bin/perl -w
use strict;
use Chart::Lines;
use Chart::Points;
use Chart::Bars;

my (@achse,@legende,@felder);
my ($datei,$diagr,$titel,$typ,
    $xachse,$yachse,$grid,$maxy,$miny);

 # 1. Zeile: Name der Ausgabedatei
 chomp($datei = <>);

 # 2. Zeile: Titel;Typ [L,P,B];Gridlines
 chomp($titel = <>);
 ($titel,$typ,$grid) = split(/\s*;\s*/,$titel);
 $typ = uc($typ);
 $typ = 'L' unless(defined($typ));
 if ($typ eq 'P')
   { $diagr = Chart::Points -> new(600,350); }
 elsif ($typ eq 'B')
   { $diagr = Chart::Bars -> new(600,350); }
 else # 'L'
```

```
  { $diagr = Chart::Lines -> new(600,350); }

# 3. Zeile: Y-Achsen-Legende, ggf. min. und max. Y-Wert
chomp($yachse = <>);
($yachse,$miny,$maxy) = split(/\s*;\s*/,$yachse);

# 4. Zeile: X-Achsen-Legende
chomp($xachse = <>);

# 5. Zeile: n X-Achsenticks
chomp($_ = <>);
@achse = split(/\s*;\s*/);
shift(@achse); # 1. Feld = Dummy
$diagr->add_dataset(@achse);

# ab 6. Zeile: Datensaetze, jeweils n pro Zeile, erster
# Datenwert = Bezeichnung
while (<>)
  {
  chomp;
  @felder = split(/\s*;\s*/);
  push(@legende, shift(@felder)); # 1. Wert
  $diagr->add_dataset(@felder);
  }

$diagr->set('title' => $titel);
$diagr->set('x_label' => $xachse);
$diagr->set('y_label' => $yachse);
$diagr->set('transparent' => 'true');
$diagr->set('legend_labels' => \@legende);
$diagr->set('y_ticks' => '10');
$diagr->set('grid_lines' => 'true') if (uc($grid) eq 'Y');
$diagr->set('min_val' => $miny) if (defined($miny));
$diagr->set('max_val' => $maxy) if (defined($maxy));
$diagr->png($datei . '.png');
```

2.1.3 Koordinatensysteme

Aber Sie müssen nicht unbedingt selbst programmieren, sondern können auf recht leistungsfähige Tools zurückgreifen. Vor der Besprechung einiger Tools möchte ich Ihnen noch ganz kurz zeigen, wie von den Programmen Messwerte, Zeitreihen etc. in das vorgegebene Raster des Bildschirms oder die Grenzen einer vorgegebenen Ratergrafik maßstabsgerecht „eingebaut“ werden. Es handelt sich mathematisch gesehen um eine lineare Abbildung der *Welt* in ein karthesisches Koordinatensystem. Bei GD wird ja von einen sehr einfachen Bildschirmkoordinatensystem ausgegangen, das dem kartesischen Koordinatensystem ähnelt und auf das sich alle Grafikprimitive stützen.

Beim kartesischen Koordinatensystem ist bekanntlich die positive X-Achse waagerecht und rechts vom Ursprung (Punkt mit den Koordinaten X = 0 und Y = 0), die positive YAchse ist senkrecht und oberhalb des Ursprungs. Weiterhin ist ein Punkt in der Ebene bei diesem Koordinatensystem durch ein Koordinatenpaar (X, Y) definiert, dessen Werte sich durch eine Projektion im rechten Winkel auf die jeweilige Koordinantenachse X bzw. Y ergeben. Der Ursprung liegt also in der linken, unteren Ecke der Zeichenfläche. Dass dies nicht immer so sein muss, zeigten die Bildschirmkoordinaten von GD, wo der Ursprung nicht in der linken, unteren Ecke, sondern in der linken, oberen Ecke liegt. Um mit GD im gewohnten System zeichnen zu können, muss man die Koordinaten auf das Kartesische Koordinatensystem transformieren. Diese Transformation von Koordinaten des Benutzers auf Koordinaten des Bildschirms soll nun betrachtet werden.

Betrachten wir zunächst das einfache Problem mit den GD-Koordinaten. Wenn das Bild die Breite W und die Höhe H besitzt, hat der Ursprung des kartesischen Koordinatensystems die Koordinaten (0,H). Es müssen also alle Punkte (X, Y) des kartesischen Koordinatensystems auf die Punkte (X, H-Y) des Bildschirms abgebildet werden, um an der gewünschten Stelle zu erscheinen. Wie lässt sich diese Tatsache verallgemeinern?

Bei fast allen Zeichengeräten ist die Bildschirm- oder Zeichenfläche begrenzt. Außerdem ist das zugrunde liegende Koordinatensystem meist recht primitiv. Die linke, obere Ecke ist meist der Koordinatenursprung, und die Skalierung ist recht einfach: Beim Bildschirm ist es die Zahl der möglichen Bildpunkte in X- und Y-Richtung, bei einer Bilddatei eine willkürliche Vorgabe des Programmierers. Bei einem elektromechanischen Zeichengerät, einem Plotter, sind es meist die Zahl der Plotterschritte

in X- und Y-Richtung (0.1, 0.05 oder 0.01 mm). Diese Skalierung entspricht aber fast nie den Wünschen des Benutzers. Will man seine Zeichnung in beliebigen Größen skalieren, zum Beispiel in Celsiusgraden, muss man die tatsächlichen Koordinaten per Koordinatentransformation auf die Bildschirmgegebenheiten abbilden.

Das Koordinatensystem soll an die Größenordnungen und Anforderungen des speziellen Problems angepasst sein (Meter, Millimeter, Grad, Sekunden, Millionen Einwohner, Jahre, etc.). Dieses *Weltkoordinatensystem* muss auf *die Koordinaten des Gerätes* abgebildet werden. Ein rechteckiges Fenster des Weltkoordinatensystems (**Window**) wird auf ein beliebiges Fenster der zur Verfügung stehenden Zeichenfläche abgebildet (**Viewport**). Bild 2.2 zeigt diesen Sachverhalt. Es gilt also:

Window: Weltkoordinaten, Skalierung

Viewport: = Gerätekoordinaten, Fenster der Zeichenfläche

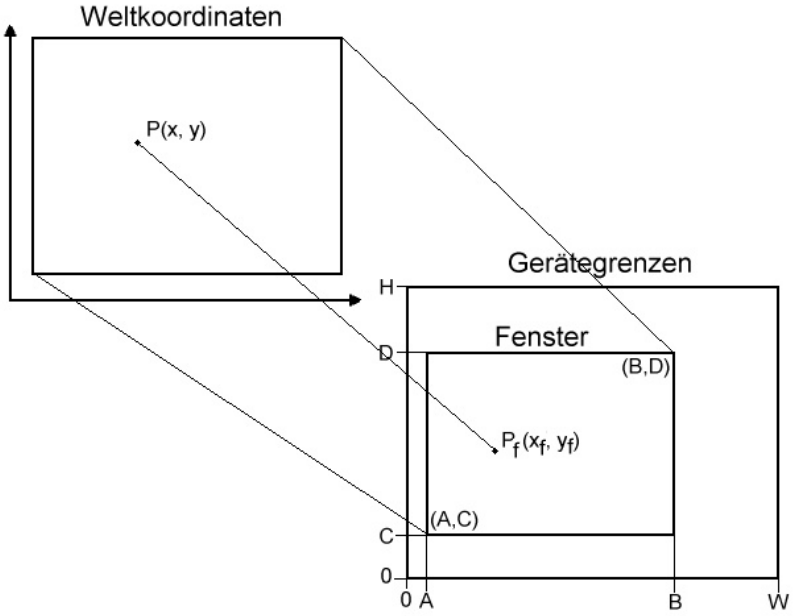

Bild 2.2: Zusammenhang zwischen Window und Viewport

Es muss also ein beliebiger Punkt in Weltkoordinaten P(X,Y) in den entsprechenden Punkt P_f(X$_f$f, Y_f) des Fensters im Gerätekoordinatensystem umgerechnet werden. Im Allgemeinen werden zwei rechteckige Bereiche aufeinander abgebildet. Für die Umrechnung brauchen wir einige Definitionen:

Die Zeichenfläche hat eine maximale Breite W und eine maximale Höhe H. Auf der Zeichenfläche wird ein Fenster definiert, das durch die Werte A,B,C, und D begrenzt ist.

Das Fenster A-B-C-D wird nun von Xmin - Xmax (Strecke zwischen A und B) und von Ymin - Ymax (Strecke zwischen C und D) skaliert. Es gilt:

$$(Xmin; Ymin) \rightarrow (A; C) \tag{2.1}$$
$$(Xmax; Ymax) \rightarrow (B; D) \tag{2.2}$$

Seien X und Y Weltkoordinaten sowie X_f und Y_f Geräte- bzw. Fensterkoordinaten, können wir folgende Betrachtungen anstellen. Dazu nehmen wir zunächst an, der linke, untere Eckpunkt des Fensters würde sich mit dem Ursprung decken. Dann gilt generell:

$$X_f = P \cdot X \tag{2.3}$$
$$Y_f = Q \cdot Y \tag{2.4}$$

wobei für P und Q gilt:

$$P = \frac{B - A}{X_{max} - X_{min}} \tag{2.5}$$
$$Q = \frac{D - C}{Y_{max} - Y_{min}} \tag{2.6}$$

Die Strecken werden also linear aufeinander abgebildet. Liegt das Fenster an beliebiger Stelle auf dem Bildschirm, kommt noch eine Translation der Ursprungskoordinaten hinzu. Die Formel lautet dann:

$$X_f = P \cdot X + S \tag{2.7}$$
$$Y_f = Q \cdot Y + T \tag{2.8}$$

wobei für S und T gilt:

$$S = \frac{A \cdot X_{max} - B \cdot X_{min}}{X_{max} - X_{min}} \tag{2.9}$$
$$T = \frac{C \cdot Y_{max} - D \cdot Y_{min}}{Y_{max} - Y_{min}} \tag{2.10}$$

Diese lineare Transformation hat nicht nur den Vorteil, dass man in Weltkoordinaten arbeiten kann, sondern auch, dass man durch Ändern der Werte von A, B, C und D das Bild beliebig skalieren kann.

Damit man im Programm auch mit den Weltkoordinaten arbeiten kann, sollte man die benötigten Grafikroutinen als eigene Funktionen definieren. Dazu reichen zunächst nur vier Routinen, die sich ihrerseits auf die Grafikprimitive (z. B. des GD-Pakets) stützen:

- `window (A, B, C, D)`
  Reservieren eines Fensters, das sich in X-Richtung von A bis B und in Y-Richtung von C bis D erstreckt (in Gerätekoordinaten, beim Bildschirm in Pixeln)
- `scale (Xmin,Xmax,Ymin,Ymax)`
  Skalieren des reservierten Fensters auch den angegebenen Bereich in Weltkoordinaten
- `moveto (X,Y)`
  Positionieren des (virtuellen) Zeichenstifts auf den Punkt (X,Y) in Weltkoordinaten, ohne eine Spur zu hinterlassen
- `drawto (X,Y,COL)`
  Zeichnen einer Linie vom augenblicklichen Standpunkt zum Punkt (X,Y) in Weltkoordinaten unter Verwendung der Farbe COL

Diese Funktionen werden an dieser Stelle nur skizziert, die Ausarbeitung bleibt Ihnen als Übung überlassen. So könnte man beispielsweise die globalen Werte in einer Verwaltungs-Struktur speichern und einen Pointer darauf jeweils als ersten Parameter übergeben.

```
void window (int a, b, c, d)
  {
  // Definieren des Zeichenfensters in Geraetekoordinaten
  // z. B. window (0, 0, 400, 200);
  // globale Konstante: W, H (maximale Maße der Zeichenfläche)
  // globale Variablen: A, B, C, D (Fensterkoordinaten)
  // A, B, C, D werden hier verändert

  A = a;
  // Randueberwachung
  if (A < 0) A = 0;
  if (A > W) A = W;

  B = b;
  // Randueberwachung
  if (B < 0) B = 0;
  if (B > W) B = W;

  C = c;
  // Randueberwachung
  if (C < 0) C = 0;
  if (C > H) C = H;

  D = d;
  if (D < 0) D = 0;
  if (D > H) D = H;
  }
```

```
int scale (float xmin, xmax, ymin, ymax)
  {
  // Skalieren des reservierten Fensters in Weltkoordinaten
  // global: int A, B, C, D (von window)
  // global: float P, Q, S, T (Skalierung)
  // P, Q, S, T werden hier veraendert
  // Rueckgabewert: 0 = OK, -1 = Fehler

  if (xmin == xmax)
    {
    printf("xmin und xmax muessen verschieden sein\n");
    return (-1);
    }
  if (ymin == ymax)
    {
    printf("ymin und ymax muessen verschieden sein\n");
    return (-1);
    }
  P = (B - A)/(xmax - xmin);
  Q = (D - C)/(ymax - ymin);
  S = (A * xmax - B * xmin)/(xmax - xmin);
  T = (C * ymax - D * ymin)/(ymax - ymin)
  return(0);
  }

void moveto (float X, Y)
  {
  // positionieren auf den Punkt (X,Y) in Weltkoordinaten
  // global: float P, Q, S, T und
  // die aktuelle Position in Geraetekoordinaten: int X0, Y0
  // X0, Y0 werden hier veraendert

  X0 = int(P * X + S);
  Y0 = int(Q * Y + T);
  }

void drawto (float X, Y; int COLOR)
  {
  // zeichnen von der Position (X0,Y0) nach (X,Y) in Weltkoordinaten
  // X0 und Y0 werden neue aktuelle Position
  // global: P, Q, S, T und
  // die aktuelle Position in Geraetekoordinaten: int X0, Y0
  // X0, Y0 werden hier veraendert
  float xu, yu;
  xu = int(P * X + S);
  yu = int(Q * Y + T);
  // Zeichne Linie von (X0,Y0) nach (xu,yu) mit Grafikprimitiv
  line(X0, Y0, xu, yu, COLOR);
  X0 = xu;
  Y0 = yu;
  }
```

2.2 Grafik-Tools

2.2.1 Gnuplot

Gnuplot ist ein plattformübergreifendes, Kommando-orientiertes, interaktives Plot-Programm, mit dem Sie mathematische Funktionen und Datensätze in zwei- und dreidimensionalen Grafiken visualisieren können. Innerhalb kurzer Zeit gelangen Sie zu ansprechenden und druckreifen Grafiken, z. B. die Überwachung der Temperatur eines Ofens, die Auswertung von Logfiles oder die Darstellung von Börsenkursen. Mit Gnuplot steht dazu ein Werkzeug zur Verfügung, das auch in Kombination mit Shellscripten und anderen Tools ungeahnte Flexibilität entfaltet. Wenn Ihnen die Shell-Programmierung nicht so liegt, können Sie auch auf grafische Frontends zugreifen, etwa „Kile“, ein Latex-Editor mit integriertem Gnuplot-Frontend, oder „Unignuplot“. Gnuplot erlaubt Visualisierungen im zwei- oder dreidimenisionalen Raum und geht auch bei extrem großen Datenmengen nicht in die Knie.

Für den Einstieg in Gnuplot existieren Tutorials und Handbücher, aber auch das Ausprobieren und Hinzulernen der Bedienung kann eine spannende Sache sein. Zudem gibt es eine Demo-Sammlung,

die Sie inspirieren könnte. Um die Arbeitsweise von Gnuplot zu verstehen, genügt ein einfaches Beispiel: Nach einer Begrüßung meldet sich Gnuplot mit seinem Prompt und Sie können Eingaben tätigen. Unser Programm soll nur zwei trigonometrische Funktionen ausgeben:

```
gnuplot> set xlabel "x"
gnuplot> set ylabel "f(x)"
gnuplot> set title "Beispiel mit Sin und Cos"
gnuplot> plot [0:(4*pi)][-1.1:+1.1] sin(2*x), cos(2*x)
```

Nach Eingabe der letzten Zeile erscheint bereits ein zweites Fenster mit der Grafikausgabe (Bild 2.3). Die Plot-Grenzen und Achsen-Skalierungen werden – sofern man nichts angibt – von Gnuplot über die autoscale-Funktion gewählt. Im Beispiel wurde der Plot-Bereich auf das x-Intervall (0, 4*pi) und das Funktionswerte-Intervall (-1.1, +1.1) eingegrenzt.

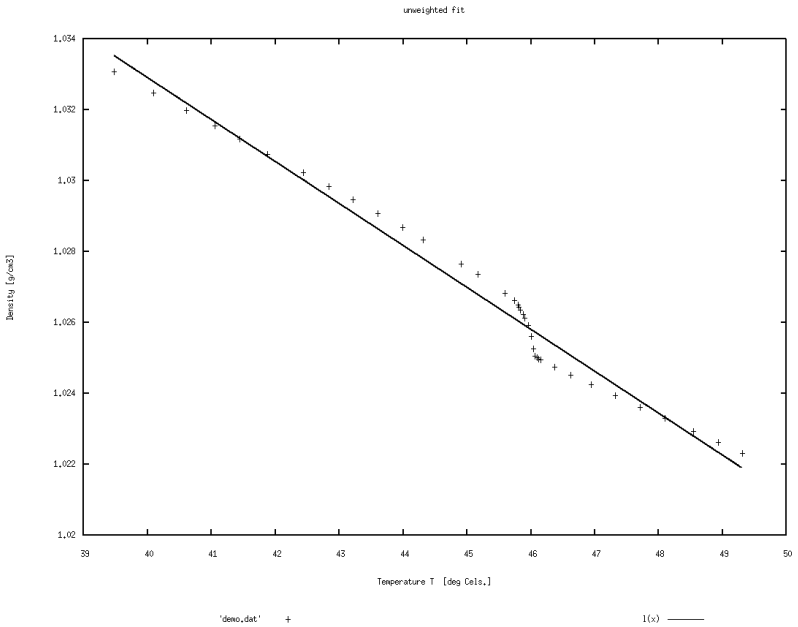

Bild 2.3: Darstellung von Messwerten einschließlich Regressionsgerade

Für die dreidimensionale Darstellung bietet Gnuplot einige spezielle Befehle, die den fiktiven Blickwinkel des Benutzers auf die 3D-Grafik beeinflussen. Zu berücksichtigen sind dabei die möglichen Rotations-Intervalle. Die Anweisung *set hidden3d* blendet verdeckte Liniensegmente aus, um den 3D-Effekt zu verstärken. Soll Gnuplot unregelmäßig verteilte Datenpunkte darstellen, empfiehlt es sich, zuerst die Lücken in der Datenpunktverteilung durch Interpolation zu schließen. Dazu dient der Befehl *set dgrid3d*, dem die Anzahl der gewünschten Intervalle in x- und y-Richtung sowie ein Gewichtungsfaktor übergeben werden.

Die aktuelle Version von Gnuplot ist in fast jeder Linux-Distribution zu finden. Weitere Informationen, das Handbuch und Versionen für alle gängigen Betriebssysteme sind unter den folgenden Homepaqge-Adressen auffindbar: *http://www.gnuplot.info*
Mehr über Gnuplot erfahren Sie in der kleinen Einführung im nächsten Kapitel.

2.2.2 LabPlot

LabPlot geht in seiner Leistung etwas über Gnuplot hinaus, ist dafür aber auch komplexer in der Bedienung. Deshalb hat es eine komfortable grafische Oberfläche. Im Übrigen dient Labplot wie Gnuplot zur Darstellung und Auswertung zweidimensionaler und dreidimensionaler Funktionen und Daten. LabPlot erlaubt Ihnen, mit mehreren Plots zu arbeiten, von denen jeder mehrere Graphen besitzen kann. Die Graphen können aus Daten oder aus Funktionen generiert werden. Das Programm kann flexibel Daten in verschiedenen Formaten lesen und schreiben.

Alle Einstellungen eines gesamten Sets von Plots lassen sich in Projektdateien speichern. Diese Projektdateien werden über die Kommandozeile, über das Dateimenü oder per Drag&Drop geladen. Jedes Objekt (Titel, Legende, Achsen, Achsenbeschriftung) kann mit der Maus gezogen werden. Es

werden alle Funktionen und Konstanten der GNU Scientific Library (GSL) unterstützt. Mit LabPlot lassen sich Oberflächen zeichnen (mit „hidden lines“) sowie Polar Plots, Ternary Plots und Tortendiagramme aus Funktionen und Datendateien erzeugen.

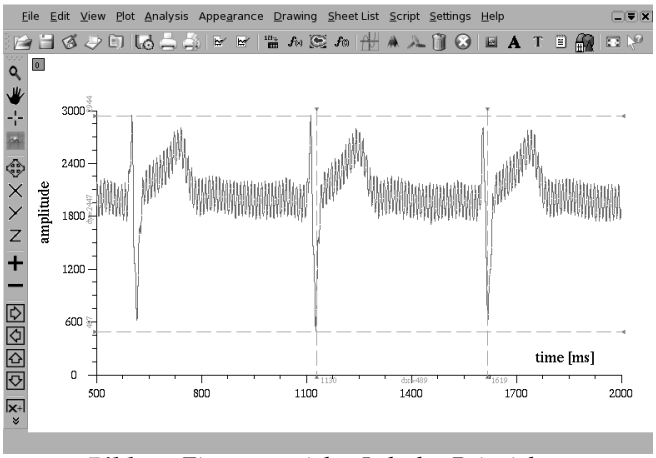

Bild 2.4: Eines von vielen Labplot-Beispielen

Die Einstellungen des Plots oder der Graphen können leicht per Menü geändert werden, und es lassen sich jederzeit zusätzliche Datensätze und Funktionen (Graphen) einfügen, die im gleichen oder einem anderen Plot dargestellt werden können. Detaillierte Dialoge für alle Einstellungen werden per Doppelklick geöffnet, und jedes Objekt lässt sich mit der Maus platzieren. Außerdem beherrscht Labplot Operationen zur Analyse von Daten und Funktionen sowie zur Mittelung, Glättung und Kürzung von Daten. Weitere Möglichkeiten sind die Komprimierung und Analyse von Daten (Periodizitäten, Spitzen, Interpolation, Integration, Regression bis zur zehnten Ordnung, nichtlineare Anpassung, Fourier-, Hankel- und Wavelet-Transformation sowie Faltung).

Die Arbeitsblätter lassen sich als Bilder (PS, EPS, SVG, PDF) exportieren oder in andere Formate konvertieren (über pstoedit or ImageMagick). Auch der Import bzw. Export von Daten in und von Datenbanken ist möglich. Die Homepage von Labplot finden Sie unter *http://labplot.sourceforge.net/index-de.html*.

2.2.3 MRTG

Die Auslastung eines Netzwerks messen und die Ergebnisse grafisch aufbereiten ist das Spezialgebiet vom Multi Router Traffic Grapher (MRTG) von Tobias Oetiker (*http://www.mrtg.org/*). Er überwacht normalerweise die Auslastung des Netzwerks, fragt Router und Switches ab und erzeugt aus den gewonnenen Daten übersichtliche Grafiken, die sich in eine Webseite einbinden lassen. Der Aufruf erfolgt in der Regel über die `crontab`. MRTG liefert jeweils getrennte Daten für unterschiedliche Zeiträume, kann also auch für die Langzeiterfassung von Daten dienen.

Viele Linux-Distributionen enthalten bereits MRTG und die zusätzlich benötigten Programme und Bibliotheken. Wenn das ausnahmsweise nicht der Fall sein sollte, stellt das manuelle Installieren kein Problem dar. Voraussetzungen sind Perl, ein C-Compiler sowie die schon erwähnte GD-Bibliothek von Thomas Boutell. Alle Komponenten laufen unter Linux, Unix und sogar Windows.

Für jedes überwachte Gerät erhält MRTG je eine Konfigurationsdatei (braucht man nur eine einzige, nimmt man */etc/mrtg.cf*. Sind mehrere Dateien geplant, legt man am besten ein Verzeichnis an und erzeugt darin bei Bedarf weitere Dateien). Die Konfigurationsdatei lässt sich auch automatisch generieren: Das Programm `cfgmaker` aus dem MRTG-Paket schreibt eine Konfigurationsdatei, mit der MRTG die Netzwerklast beobachtet. Das Programm `indexmaker` erzeugt eine HTML-Startseite, in der alle überwachten Geräte verzeichnet sind. Im *contrib*-Verzeichnis der MRTG-Distribution finden Sie viele fertige Module, die zum einen die Leistungsfähigkeit von MRTG in seinen Erweiterungen unter Beweis stellen.

MRTG kann jedoch nicht nur Router- oder Rechnerdaten grafisch aufbereiten, sondern jede beliebige Zeitreihe von ein oder zwei unabhängigen Werten darstellen. Damit eignet es sich vorzüglich, um Messwerte zu sammeln und grafisch zu präsentieren. Statt SNMP für die Datensammlung zu benutzen, lassen sich eigene Scripte und Programme in MRTG einbinden, die extern Daten sammeln und

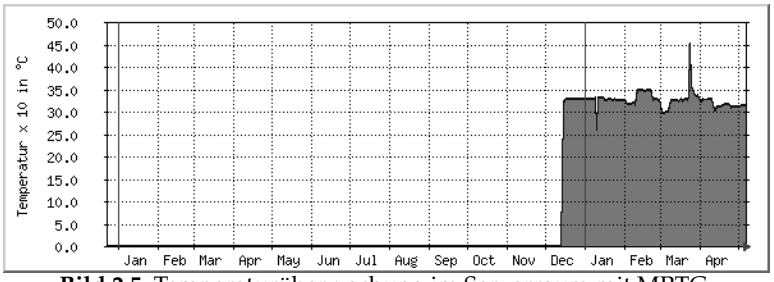

Bild 2.5: Temperaturüberwachung im Serverraum mit MRTG

an MRTG übergeben. Das Beispiel in Bild 2.5 zeigt den Temperaturverlauf in unserem Serverraum (jährlicher Verlauf). Man erkennt deutlich einen Ausfall der Klimaanlage Ende März.

3

Visualisierung mit Gnuplot

Gnuplot ist ein kommandozeilenorientiertes, interaktives wissenschaftliches Plotprogramm. Es ist klein, mächtig und bietet alle Möglichkeiten Daten schnell darzustellen. Gnuplot ist kostenlos (*http://www.gnuplot.info*). Mit Gnuplot lassen sich Funktionen und Daten (Messwerte) in einem zweidimensionalen karthesischen Koordinatensystem oder in einem 3D-Raum darstellen. Flächen werden dabei als Netzgittermodell im dreidimensionalen Raum dargestellt oder als Kontur in die XY-Ebene projiziert. Für 2D-Darstellungen stehen zahlreiche Arten der Darstellung (Styles) zur Verfügung, z. B. Linien, Punkte, punktierte Linien, Balken etc. Die Kurven und Achsen können beschriftet werden, Markierung, Überschriften, Datums-und Zeitangaben usw. sind ebenfalls möglich. Die primäre Anwendung besteht darin, Werte darzustellen, die man aus Berechnungen, Messungen u. a. erhalten hat. Daneben lassen sich beliebige Funktionen (Polynome, trigonometrische Funktionen etc.) plotten werden und Kuvenverläufe interpolieren, beispielsweise Messwerte durch Polynome annähern. Zusätzlich zum karthesischen Koordinatensystem kennt Gnuplot auch polare Koordinatensysteme.

Im einzelnen bietet Gnuplot folgenden Funktionsumfang:

- Zeichnen zweiudimensionaler Funktionen f(x) in unterschiedlichen Stilarten (Punkte, Linien, Fehlerbalken, ...)
- Zeichnen dreidimensionaler Funktionen g(x,y) in unterschiedlichen Stilarten (Kontur, Oberfläche, Hinzufügen von Gittern)
- Verwenden der mitgelieferten mathematischen Funktionen wie abs(x), sqrt(x) und von einfachen benutzerdefinierten Funktionen
- Verwenden komplexer Daten (x,y) und Funktionen
- Darstellen von zwei- und dreidomensionalen Daten(-Dateien)
- Steuerungsmöglichkeit über Gnuplot-Kommandodateien (Batch-Betrieb)
- Exportieren von Abbildungen in diversen Grafik-Formaten
- Hinzufügen von Gestaltungselementen (Titel, Achsenbeschriftungen, Achseneinteilung, Legende, Pfeile, ...)

Beim Start von Gnuplot sucht das Programm zunächst nach einer Initialisierungsdatei `.gnuplot` im Heimat-Verzeichnis. Dort können Gnuplot-Befehle untergebracht werden, die bei jedem Programmstart zuerst ausgeführt werden sollen. Auf diese Weise lassen sich die Grundeinstellungen der auszugebenden Grafik festlegen und es müssen nur noch die aktuellen Daten eingegeben werden. Weil Gnuplot seine Befehle jederzeit auch aus einer Pipe bzw. Datei lesen kann, steht einer vollautomatischen Erzeugung von Datenauswertungen nichts im Wege.

In diesem Kapitel werden hauptsächlich die Aspekte der Visualisierung von Messwerten berücksichtigt. Daherkönnen leider auch viele schöne und mächtige Funktionen von Gnuplot nicht behandelt werden.

3.1 Einführung

Gnuplot ist, wie gesagt kommandozeilenorientiert. Man startet es einfach im Shell-Fernster mit dem Kommando `gnuplot`. Dann kommt etwas Text (Gnuplot-Version, Programmierer, Webseite usw.) und schliesslich der Gnuplot-Prompt `gnuplot>`.

Nun kann das Programm interaktiv bedient werden. Zwei wichtige Kommandos sollten Sie sich gleich merken: `quit` zum Beenden und `help`, um die Hilfe abzurufen. Ist eine Eingabe erfolgt, wird sie sofort ausgeführt, was man entweder daran merkt, dass sich etwas tut (z. B. bei einem Plotbefehl) oder es werden Variablen und Definitionen gespeichert, die sich auf nachfolgende Plotbefehle auswirken. Das kann man mit Hilfe der eingebauten Funktionen sofort ausprobieren:

```
gnuplot> f(x) = sin(x)
gnuplot> plot f(x)
```

Im Prinzip hätte ein Befehl, `plot sin(x)`, ausgereicht. Das Ergebnis zeigt Bild 3.1.

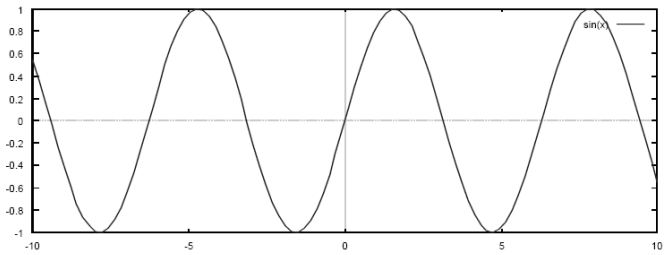

Bild 3.1: Der erste Gnuplot-Versuch

Es kann sinnvoll sein, die Befehlsfolgen nicht immer wieder neu einzugeben, sondern in einer Script-Datei (Textdatei) zu speichern, die dann mit dem Befehl `load Datei` aufgerufen werden kann oder gleich auf der Kommandozeile an Gnuplot übergeben wird. So kann man eine Script-Datei auch immer wieder erweitern oder unterschiedliche Daten mit identischer Formatierung darzustellen. So kann beispielsweise der `plot`-Befehl um die Achsengrenzen erweitert werden:

```
gnuplot> f(x) = sin(x)
gnuplot> plot [-10:10] [-1:1] f(x)
```

In der ersten eckigen Klammer stehen `xmin:xmax` und in der zweiten `ymin:ymax`. Man kann die zweite Klammer mit den Y-Grenzen auch weglassen, dann skaliert Gnuplot automatisch.

Um mehrere Funktionen zu plotten gibt man beim `plot`-Befehl einfach mehrere durch Komma getrennte Funktionen an, z. B. `plot [0:5] sin(x), exp(x)`. Gnuplot kennt eine recht ansehnliche Menge von Funktionen, darunter cos(x), sin(x), cosh(x), sinh(x), exp(x), sqrt(x), pi usw. Natürlich lassen sich auch neue Funktionen definieren:

```
gnuplot> f(x) = x**2 - x
gnuplot> plot f(x)
```

Es gibt etliche Standard-Einstellungen über den Stil der Abbildung etwa, dass die erste Grafik in rot gezeichnet wird. Angezeigt werden die gültigen Einstellungen mit dem Befehl `show all`. Diese Grundeinstellungen stehen in einer Datei namens *`.gnuplot`*. Diese Defaultwerte kann man natürlich überschreiben, indem man die Datei bearbeitet. Anderen Script-Dateien können per Befehl erzeugt werden (`save Datei` und, wie schon erwähnt, mittels `load Datei` wieder geladen werden. Gnuplot speichert die benutzerdefinierten Funktionen, Variablen usw. in einer ASCII-Datei.

In allen Script-Dateien, aber auch in den unten besprochenen Datendateien sind Kommentare erlaubt, die wie bei der Shell oder Perl mit einem „#“ beginnen.

Der prinzipielle Befehl für Änderungen an Achsen, Label, Wertebereiche etc. lautet `set`. Eine Änderung wirkt solange, bis man die Einstellungen wieder neu festlegt. Mit dem Kommando `reset` kehrt Gnuplot zu den Standardeinstellungen zurück. Statt die Skalengrenzen beim `plot`-Befehl festzulegen, kann man sie auch global definieren:

```
gnuplot> set xrange [-10:10]
gnuplot> set yrange [-250:250]
gnuplot> a = 0.8
gnuplot> b = -2.5
gnuplot> c = -30
gnuplot> f(x) = a*x**3 + b*x**2 + c*x - 15
gnuplot> plot f(x)
```

Zuerst werden die darzustellenden Achsenbereiche festgelegt und dann einige Konstanten definiert. Diese werden in der Funktionsdefinition verwendet. Der Plot-Befehl stellt die Funktion dar. Fehlt noch die Beschriftung in Form einer Bildüberschrift und der Achsenbeschriftung. Die kann man aber nachholen:

```
gnuplot> set xrange [-10:10]
gnuplot> set yrange [-250:250]
gnuplot> set title "Ein Testplot"
gnuplot> set xlabel "X-Achse"
gnuplot> set ylabel "Y-Achse"
gnuplot> a = 0.8
gnuplot> b = -2.5
gnuplot> c = -30
gnuplot> f(x) = a*x**3 + b*x**2 + c*x - 15
gnuplot> plot f(x)
```

Langsam lohnt sich schon eine Script-Datei, insbesondere, wenn Sie mittels `xtics` und `ytics` auch noch den Abstand der Achsen-Teilstriche festlegen oder ein Koordinatengitter mit `set grid` erzeugen. Bei kleinen Änderungen ist es übrigens günstiger, den Befehl `replot` zu verwenden, der einfach den letzten Plot-Befehl wiederholt. Das ist insbesondere dann praktisch, wenn man dem Plot-Befehl etliche Parameter mitgegeben hat (über die weiter unten gesprochen wird).

Zum Anschauen der Grafiken ist der Bildschirm ja recht schön, aber für die Druckausgabe oder das automatische Generieren von Grafiken für eine Webseite eignet sich das Terminal nicht. Gnuplot beherrscht aber zahlreichen Ausgabemöglichkeiten. Zum Ausdrucken von Plots werden die Befehle `set output` und `set terminal` benötigt. Mit letzterem Befehl lässt sich bei Gnuplot die Grafikschnittstelle wählen (Default: X11). Andere Optionen sind unter anderem `epson`, `pcl5`, `postscript`, `png` und `latex`. Ein weiteres, sehr leistungsfähiges Gnuplot-Terminal ist `svg`, das SVG-Dateien erzeugt. Das `svg`-Terminal beherrscht schon die Option „transparent“ und ermöglicht so einige nette Effekte.

In Verbindung mit `set output` kann eine Druckdatei erzeugt oder direkt gedruckt werden. Beispielsweise sorgt die Einstellung `set terminal postscript` für eine Ausgabe im Postscript-Format. In diesem Fall sollten Sie die Ausgabe mit `set output` in eine Datei lenken, die sich auf einem Postscript-Drucker ausgeben oder in ein LaTeX-Dokument einbinden lässt:

```
gnuplot> set xrange [-10:10]
gnuplot> set yrange [-250:250]
gnuplot> set title "Ein Testplot"
gnuplot> set xlabel "X-Achse"
gnuplot> set ylabel "Y-Achse"
gnuplot> a = 0.8
gnuplot> b = -2.5
gnuplot> c = -30
gnuplot> f(x) = a*x**3 + b*x**2 + c*x - 15
gnuplot> plot f(x)
gnuplot> set terminal postscript
gnuplot> set output "grafik.ps"
gnuplot> replot
gnuplot> quit
```

Das Ergebnis sehen Sie in Bild 3.2. Welche Parameter Sie dem `set terminal`-Befehl noch mitgeben können, beschreibt die Gnuplot-Dokumentation. Beispielsweise kann die Orientierung angegeben werden (`landscape` oder `portrait`, aber auch Farben oder Schriftart und -grösse können hier noch festgelegt werden. Wenn alle Stricke reißen, können Sie auch einen Screenshot speichern oder über die Zwischenablage in ein anderes Programm einfügen. In diesem Fall ist die Qaualität aber eher bescheiden.

Die neueren Versionen von Gnuplot beherrschen nun auch UTF-8 und können nun auch Umlaute sauber darstellen. Gnuplot ermöglicht neben der Darstellung von Funktionen zusätzlich die Darstellung von 2D-Kurven und 3D-Flächen als parametrische Kurven, worauf an dieser Stelle ebenso wenig eingegangen wird, wie auf die Verwenden von kartesischen, Zylinder- und Kugelkoordinaten.

3.2 Darstellung von Daten-Dateien

Für die Datenauswertung und -visualisierung ist das Plotten von Funktionen nur dann interessant, wenn die Funktion den Daten gegenübergestellt wird, etwa bei einer interpolierenden Funktion. Darum wenden wir uns nun dem Plotten von Daten zu.

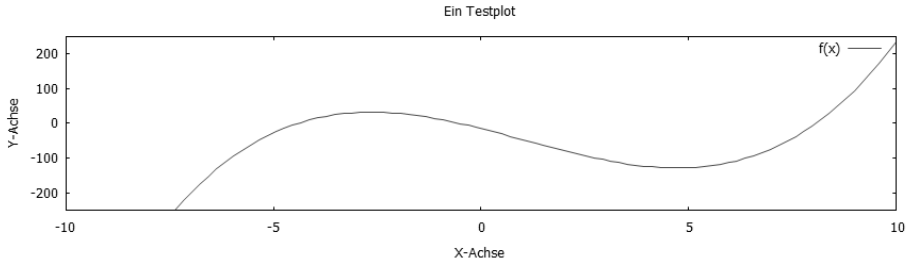

Bild 3.2: Ein Funktionsplot mit Achsenbeschriftung und Überschrift

Meist liegen die zu plottenden Daten in einer (Text-)Datei vor. Auch in einer sind Kommentarzeilen erlaubt, die mit einem „#“ beginnen. Die einzelnen Werte werden durch mindestens ein Leerzeichen voneinander getrennt. Eine besondere Bedeutung haben hingegen *Leerzeilen*. Eine Leerzeile unterbricht die Verbindungslinie zwischen den Datenpunkten (sofern sie durch Linien verbunden sind), zwei Leerzeilen signalisieren Gnuplot, dass ein neuer Datensatz beginnt. Im Übrigen dürfen die Daten frei formatiert werden.

Die Datei enthält meist ein X-Y-Paar pro Zeile, also zwei Spalten. Es dürfen auch mehr Spalten sein, aber dann muss angegeben werden, welche Spalten für die X- und Y-Werte zuständig sind. Man kann den X-Wert sogar weglassen, dann zählt Gnuplot den X-Wert automtatisch von 0 beginnend hoch. Die folgende Beispiel-Datei soll geplottet werden:

```
1 5
2 4
3 7
...
```

Um diese Daten zu plotten, wird anstelle einer Funktion ein Dateiname beim `plot`-Befehl angegeben, z. B. `plot 'daten.txt'`. Per Default markiert Gnuplot nur die Datenpunkte mit kleinen Kreuzen. Wenn Sie etwas anderes wollen, können Sie aus einer Vielzahl von Möglichkeiten wählen, indem Sie den Befehl erweitern:

```
gnuplot> plot 'daten.txt' with <style>
```

Natürlich sind unterschiedliche Linientypen möglich. Es gibt u. a. folgende *style*-Attribute des *plot*-Befehls: *lines*, *points*, *linespoints*, *dots*, *impulses*, *steps*, *fsteps*, *histeps*, *boxes*, *errorbars*, *xerrorbars*, *yerrorbars*, *xyerrorbars*, *boxerrorbars*, *boxxyerrorbars*, *vector*, *financebars*, *candlesticks*. Die Syntax lautet immer *plot „Datei“ with Style*. Ebenso sind bereits Statistik- und Glättungsfunktionen eingebaut.

In der folgenden Liste sind einige mögliche Werte von `style` aufgeführt:

lines: Verbindung der Punkte mit Linien (Standard für Funktionen)

points: Verbindung der Punkte mit Kreuzen (Standard für Daten aus Dateien)

linespoints: Verbindung der Punkte mit unterbrochenen Linien

dots: Verbindung der Punkte mit Punkten

boxes: Verbindung der Punkte mit Balken, z. B. für Histogramme; die X-Koordinate liegt immer auf den Mittelpunkt der Box

steps: Verbindung der Punkte mit Stufen (wie Balken, nur ohne senkrechte Striche)

impulses: senkrechte Linie vom Punkt zur X-Achse

Man kann Linienstärke, Linienart, Punktgröße und Punktart festlegen:

linetype (lt): Angabe des Linienstils. In der Regel die Farbe und, falls verwendet, den entsprechenden Punkt. Die Linienstile lassen sich mit dem Befehl `test` anzeigen.

linewidth (lw): Stärke der Line; je höher dieser Wert ist, desto dicker wird der Strich.

pointtype (pt): Angabe des Stils eines Punktes. Wie die Punkte aussehen, lässt sich auch hier mittels `test` anzeigen.

pointsize (ps): Größe des Punktes; je höher der Wert ist, desto größer ist der Punkt.

Beispiel:

```
gnuplot> plot 'daten.txt' with linespoints \
         linetype 2 linewidth 3 pointtype 1 pointsize 2
```

Im obigen Listing wurde die Zeile geteilt, indem als letztes Zeichen vor dem Zeilenende ein Backslash eingegeben wurde.

Weiterhin kann die Kurve eine Legende erhalten, indem man `title "Legende"'` hinzufügt. der Parameter `notitle` unterdrückt die Standard-Legende. Bei mehreren Kurven in einer Grafik fügt man die Angaben zur jeweils nächsten Kurve hinten an. Hat die Datendatei mehr als zwei Spalten, können mittels `using x:y` die Spaltennummer für die X- und Y-Achse ausgewählt werden. Bei einer Datei mit nur zwei Spalten kann der Parameter `using 1.2` natürlich entfallen. Das folgende Beispiel beruht auf der Datendatei

```
# x, y, delta_y
0.0 2.1 0.15
0.5 3.0 0.15
1.0 3.6 0.2
1.5 4.3 0.2
2.0 4.3 0.2
2.5 3.4 0.15
3.0 2.0 0.15
```

Der folgende Befehl plottet die Daten der Datei zweimal: zuerst nur die Fehlergrenzen (errorbars), danach werden alle Punkte durch eine Linie verbunden.

```
gnuplot> plot "daten.txt" title "Errorbars" with errorbars linewidth 2,\
         "daten.txt" title "Line" with lines linewidth 2
```

Der Typ `errorbars` benötigt nicht nur zwei, sondern drei Spalten in der Datendatei, wobei die Dritte Spalte die maximale Abweichung vom Y-Wert angibt. Gnuplot macht dann alles Weitere. Das Ergebnis zeigt Bild 3.3.

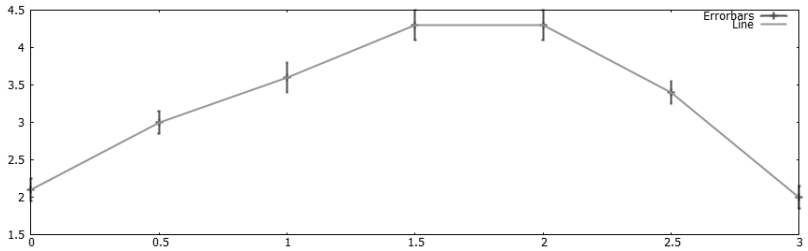

Bild 3.3: Eine Grafik mit Linie und Fehlergrenzen

Manchmal hat man es mit speziellen Datenformaten zu tun. Wenn z. B. der tageweise Zeitverlauf einer Messung dargestellt werden soll, muss Gnuplot eine Datums- oder Zeitangabe als X-Wert erkennen und nicht als Zahl interpretieren. Hier ist es hilfreich, wenn Sie sich mit dem UNIX-Kommando `date` auskenne, denn das Format und die Formatierungszeichen sind bei Gnuplot dieselben. Angenommen Ihr privates Computermuseum hat folgende Besucherzahlen:

```
10.09.12 15
11.09.12 18
12.09.12 12
13.09.12 15
14.09.12 19
15.09.12 34
16.09.12 35
```

Jede dieser Zeilen besteht also aus Datum und Besucherzahl. Um Gnuplot mitzuteilen, dass Sie an Daten mit Zeitwerten interessiert sind, geben Sie den Befehl `set xdata time` ein. Nun muss noch das Format festgelegt werden, was mit `set timefmt "..."` erfolgt. Die Formatangabe entspricht hierbei der des `date`-Kommandos. Für die obige Beispieldatei lautet der Gnuplot-Befehl `set timefmt '%d.%m.%y'`.

Schliesslich sollten Sie noch angeben, welche Zeitinformationen als Achsenbeschriftung verwendet werden sollen. Dies wird mit dem Befehl `set format x "..."` erreicht. Für das Format „Tag/Monat“ schreiben Sie `set format x "%d/%m"` oder für „Jahr.Monat.Tag“ `set format x "%y.%m%.%d"`. Alles zusamme ergibt folgende Gnuplot-Befehle:

```
gnuplot> set xdata time
gnuplot> set timefmt "%d.%m.%y"
gnuplot> set format x "%d-%m"
gnuplot> set ylabel "Besucher"
gnuplot> set title "Besucherstatistik vom 10.9.2012 bis 16.9.2012"
gnuplot> set yrange [0:50]
gnuplot> set style fill pattern 4
gnuplot> plot "besucher.txt" using (timecolumn(1)):2 title "Besucher" with boxes
```

Die resultierende Grafik zeigt Bild 3.4 .

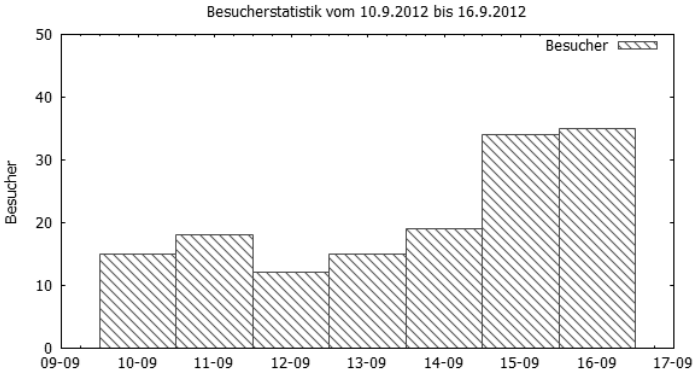

Bild 3.4: Balkendiagramm der Besucherzahlen

Falls es wahrscheinlich ist, dass sich die Messgrößen durch eine Kurve interpolieren lassen, können Sie ein Polynom finden, das sich möglichst genau an die Messpunkte anschmiegt. Das Verfahren nennt sich „least squares fit = Methode der kleinsten Quadrate“ und besteht darin, eine Kurve zu finden, bei der die Summe der Quadrate der Abweichungen zwischen Kurve und einzelnen Messpunkten minimal ist.

Dazu muss eine Funktion vorgegeben werden, z. B. $f(x) = a * x**3 + b * x**2 + c * x + d$, also ein Polynom dritten Grades. Gnuplot versucht nun durch Iteration die besten Werte für a, b, c und d zu finden, für die das oben genannte Kriterium gilt. Der Befehl dazu heisst `fit`. Anschließend können Sie noch Datenpunkte und Kurve übereinander plotten:

```
gnuplot> set xrange [0:6]
gnuplot> set yrange [0:15]
gnuplot> f(x) = a * x**3 + b * x**2 + c*x + d
gnuplot> fit f(x) 'dat.txt' via a,b,c,d

   ... (ganz viel Output)

a               = 0.0160062
b               = 0.155478
c               = 0.911267
d               = 2.31119

   ... (noch mehr Output)

gnuplot> plot 'dat.txt' with points, f(x)
```

Wie Bild 3.5 zeigt, passt die Kurve ganz gut zu den Daten.

Manchmal möchte man Funktionen zeichnen, die nicht in den üblichen kartesischen Koordinaten angegeben sind. Hierzu wird der Befehl `set parametric` verwendet. Die Variablenbezeichnungen sind `t` für zweidimensionale bzw. `u` und `v` für dreidimensionale Funktionen. Ein Einheitskreis kann folgendermaßen gezeichnet werden:

```
set parametric
plot cos(t), sin(t)
```

Es lassen sich auch ausgefüllte Flächen mit Gnuplot darstellen. Dies geschieht einfach über die `plot`-Option `with filledcurve`. Soll der Kreis des letzten Beispiels ausgefüllt werden, schreibt man:

```
set parametric
plot cos(t),sin(t) with filledcurve
```

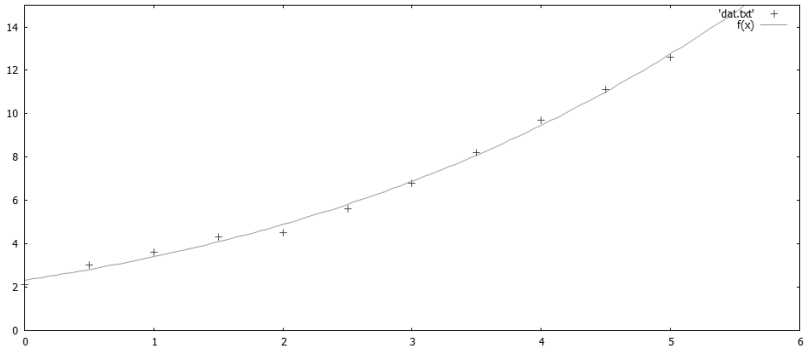

Bild 3.5: Messwerte mit interpolierendem Polynom

Gefüllten Flächen haben stets dieselbe Farbe wie die Funktion, von der sie begrenzt werden. Es kann vorkommen, dass das Programm die Funktionen invertiert ausfüllt. In diesem Fall kann die zweite Grenze manuell angeben werden. So stellt z. B. der Befehl `plot x**2 with filledcurve y1=0` eine Fläche zwischen der Parabel und der y-Achse dar, wie in Bild 3.6 gezeigt.

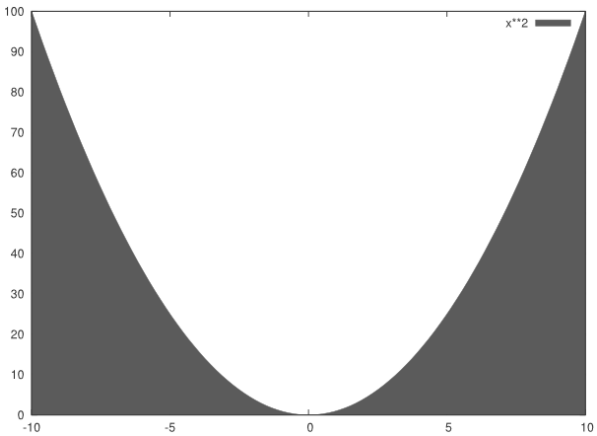

Bild 3.6: Gefüllte Kurven mit Gnuplot

3.3 Gnuplot per Script ausführen

Um nicht jedesmal ein und dieselben Befehle einzugeben, wenn man verschiedene Daten visualisieren möchte, kann man, wie schon erwähnt, gnuplot auch durch Scripte steuern. Dazu legt man beispielsweise eine Datei „plot.gp“ mit folgendem Inhalt an:

```
#!/usr/bin/gnuplot
set term postscript
set output "ergebnis.ps"
set title "Ein Testplot"
set xlabel "X-Achse"
set ylabel "Y-Achse"
plot "daten.dat" notitle using 1:2 with lines
```

Dann setzt man die Ausführen-Berechtigung für die Datei (`chmod +x plot.gnu`) und ab da kann man die Datei einfach ausführen wie beispielsweise ein Shellscript (dabei liest die Shell die erste Zeile und weiss dann, dass `gnuplot` mit der Datei als Eingabe gestartet werden muss.

Übrigens versucht gnuplot beim Speichern genau das schon vorzubereiten und fügt eine entsprechende Zeile ein. Sie sollten aber überprüfen, ob der dort angegebene Pfad stimmt und die Zeile ggf. anpassen.

Wird Gnuplot mit einer Datei als Parameter aufgerufen, führt es die angegebene Datei bis zur letzten Zeile aus und beendet sich dann sofort. Wenn im Batch-Betrieb Grafikdateien erzeugt werden sollen, ist das ja auch in Ordnung. Wenn Sie dagegen das Ergebnis ansehen wollen, sehen Sie nichts, weil Gnuplot ja sofort beendet wird. Sie könnten in der letzten Zeile der entsprechenden Batch-Datei den

`pause`-Befehl verwenden. Mit einem Wert von -1 wartet Gnuplot sogar, bis Sie eine Taste drücken. Einfacher ist es jedoch, Gnuplot mit der Option `-persist` aufzurufen. Dann wird ebenfalls auf einen Tastendruck gewartet.

Innerhalb eines Shell-Scripts kann man Gnuplot auch ohne extra Steuerdatei aufrufen, wenn man ein Here-Dokument verwendet, zum Beispiel:

```
# demonstriert Gnuplot mit Here-Dokument
gnuplot -persist <<PLOTEND
set xrange [0:6]
set yrange [-20:40]
set xlabel "Tag"
set ylabel "Grad/Celcius"
set data style linespoints
set title "Temperatur-Daten 2012"
plot "temp.dat" title "2012"
quit
PLOTEND
echo "Done ..."
```

3.4 Dreidimensionale Darstellung

Ddimensionale Funktionen f(x,y) lassen sich als Flächen im Raum darstellen. Dazu müssen die Achsenbereiche von x-, y- und z-Achse sowie Blickwinkel und -höhe (Azimuth und Elevation) mit `set view` festgelegt werden. Nachdem die Funktion definiert wurde, kann sie direkt dem Befehl `splot` übergeben werden. Da hier keine diskreten Werte vorliegen, muss noch angegeben werden, wie dicht die Gitternetzlinien gesetzt werden sollen (`set isosamples`). Um den 3D-Effekt zu verstärken, können verdeckte Liniensegmente ausgeblendet werden (`set hidden3D`). Die Befehle dazu lauten:

```
gnuplot> set hidden3d
gnuplot> set isosamples 40
gnuplot> set xrange [-2.5:2.5]
gnuplot> set yrange [-2.5:2.5]
gnuplot> set zrange [0.0:5.0]
gnuplot> set xtics 0.5
gnuplot> set ytics 1.0
gnuplot> set ztics 0.5
gnuplot> set view 40,50,1.0,1.0
gnuplot> f(x,y)=(x**2+4*y**2)*exp(1-(x**2+y**2))
gnuplot> splot f(x,y)
```

Bild 3.7 zeigt das Ergebnis. Experimentieren Sie ruhig mal mit dem Befehl `set view`, indem Sie den Befehl modifizieren und dann `replot` eingeben. So bekommen Sie ein Gefühl für die dritte Dimension.

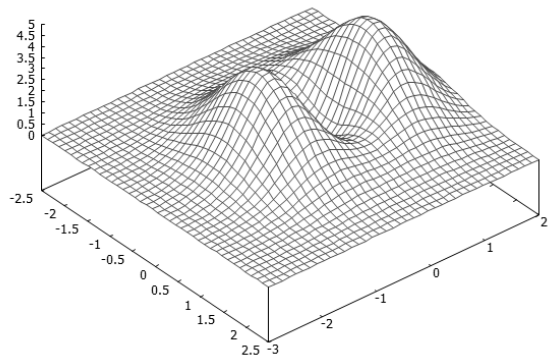

Bild 3.7: Dreidimensionale Funktionen mit Gnuplot

Wie schon bei den zweidimensionalen Anwendungen,lassen sich auch Daten dreimensional darstellen. Von der Besucherstatistik unseres Computermuseums wurden die Besucher wochenweise aufsummiert und nun kann man die Statistik des letzten halben Jahres visualisieren. Die Datendatei hat jeweils drei Werte pro Zeile:

```
1  7  44
2  7  32
3  7  41
4  7  31
  ...

4 11  58
1 12  43
2 12  43
3 12  53
4 12  48
```

Die Gnuplot-Befehle lauten folgendermaßen, um das Bild 3.8 zu erzeugen. Standardmäßig würden nur `points` verwendet:

```
gnuplot> set xrange [1:4]
gnuplot> set yrange [7:12]
gnuplot> set zrange [0:100]
gnuplot> set xlabel "Woche"
gnuplot> set ylabel "Monat"
gnuplot> set zlabel "Besucher"
gnuplot> set view 80,97,1,1
gnuplot> splot "besucher2.txt" title "Besucher" with impuls linewidth 10
```

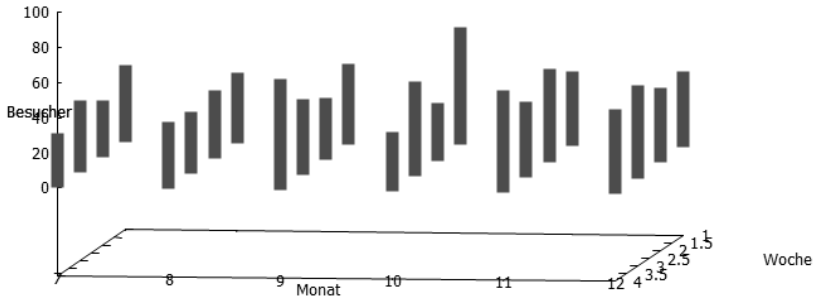

Bild 3.8: Dreidimensionale Darstellung von Messwerten mit Gnuplot

Jetzt wird die Befehlsfolge geändert bzw. erweitert. Der Datensatz mit seinen unregelmäßig verteilten Datenpunkten (scattered data) soll als Netzgitter-Oberfläche dargestellt werden. Dafür müssen zunächst durch Interpolation die Lücken gefüllt und ein äquidistantes Netz von Datenpunkten geschaffen werden (sogenanntes gridding). Jeder Datenpunkt trägt in Bezug auf einen Gitterpunkt zu dessen Wert bei und zwar gewichtet nach seiner Entfernung zu diesem Punkt. Multipliziert wird das noch mit einem Gewichtungsfaktor. Das Gridding des Datensatz erfolgt mit dem Befehl `dgrid3d`. Der Befehl benötigt die Anzahl von Intervallen in X- und Y-Richtung, die das Gitter aufspannen und einen Gewichtungsfaktor. In Beispiel habe ich einfach jeweils 10 Stützpunkte verwendet, was zu den Werten 40 (4 Wochen * 10) und 60 (6 Monate * 10) führt. Der Gewichtungsfaktor 5 sorgt dafür, dass man überhaupt Unterschiede sieht.

```
gnuplot> set xrange [1:4]
gnuplot> set yrange [7:12]
gnuplot> set zrange [0:100]
gnuplot> set xlabel "Woche"
gnuplot> set ylabel "Monat"
gnuplot> set zlabel "Besucher"
gnuplot> set view 80,97,1,1
gnuplot> set dgrid3d 40,60,5
gnuplot> splot "besucher2.txt" with lines
```

Geplottet wird mit Linien, wie Bild 3.9 zeigt. Beim Gridding sollte man sich die Tatsache vor Augen halten, das hier aus ein paar verstreuten Datenpunkten eine komplette Oberfläche errechnet wird. Das Interpolieren der Lücken beruht ja nur auf der Annahme, dass die Punkte in der Umgebung von Messwerten sich so ähnlich wie die Messwerte selbst verhalten. Der Grid kann natürlich nur so gut sein, wie die Datendichte und Datenqualität es zulässt. Auch wenn man aus wenigen Datenpunkten optisch viel zaubern kann (wie Bild 3.9 beweist), ist der Mut zur Lücke oft korrekter.

Durch das Gewichtungsverfahren, die Dichte des Grids und die Gewichtungsfaktoren kann man beeinflussen, wie groß der Einfluss eines Messwertes auf seine Umgebung ist: Je weiter der Radius um

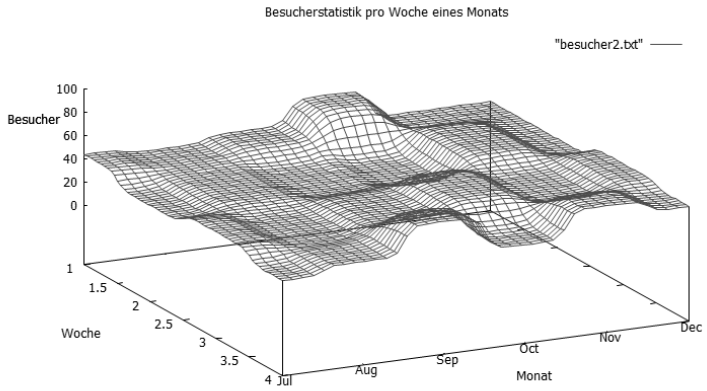

Bild 3.9: Dreidimensionale Glättung der Messwerte

einen Gitterpunkt ist, innerhalb dessen Messwerte in die Berechnug des Knotenpunkts noch spürbar eingehen, desto weniger spielen lokale Extremwerte eine Rolle (und das Ergebnis wirkt optisch glatter).

Auch parametrische Kuven können dreidimensional dargestellt werden. Ganz einfach lässt sich beispielsweise ein Zylinder durch folgenden Aufruf erzeugen:

```
set parametric
plot cos(u), sin(u), v
```

Die aktuelle Version von Gnuplot ist in fast jeder Linux-Distribution zu finden. Außerdem gibt es Versionen für Windows und für Mac OS/X. Weitere Informationen, das Handbuch und Versionen für alle gängigen Betriebssysteme sind unter den folgenden Adressen auffindbar:

Homepage: *http://www.gnuplot.info*
Tutorial: *http://www.duke.edu/hpgavin/gnuplot.html*
Tutorial: *http://www3.physik.uni-stuttgart.de/studium/praktika/ap/pdf_dateien/Allgemeines/Beschreibung-Gnuplot.pdf*
Kurs: *http://userpage.fu-berlin.de/~voelker/gnuplotkurs/gnuplotkurs.html*
Manual: *http://www.tu-chemnitz.de/urz/anwendungen/grafik/gnuplotdoc.html*
FAQ: *http://www.ucc.ie/gnuplot/gnuplot-faq.html*
Gnuplot-Tricks: *http://gnuplot-tricks.blogspot.de/*

Stichwortverzeichnis