ÜBER DIE MOLER-MORRISONSCHE BERECHNUNG DER VEKTORLÄNGE

Oktober 1989

C. B. Moler und D. R. Morrison haben eine atemberaubende Methode zur Errechnung von $\sqrt{(x^2+y^2)}$, der "pythagoreischen Summe", vorgeschlagen: "IBM J. Res. Develop.", **27**, 6, 577-581.

Ich gebe eine (vorläufige) Variante in C für "double":

```
double fleng(double x,double y)
{
double p, q, r;
  p = fabs(x); q = fabs(y); if (p < q) { r = p; p = q; q = r; }
  /* that is, p = max(|x|,|y|), q = min(|x|,|y|) */
  while (q) { r = q/p; r *= r; r /= r+4; p *= 2*r+1; q *= r; }
  return(p);
}
```

Freilich können, statt p und q, x und y benutzt werden.

Weiter unten, unter "Varianten", werden noch Verbesserungen angegeben.

Zur Analyse

Zeichen ohne Sternchen bedeuten die Werte vor einem Iterationsschritt, solche mit Sternchen bedeuten diejenigen nach ihm.

$p^* = p(4p^2+3q^2)/(4p^2+q^2)$, $q^* = q.q^2(4p^2+q^2)$.

Hieraus berechnet man unmittelbar

$p^{*2}+q^{*2} = p^2+q^2$,

d. h., die Quadratsumme, somit auch die Wurzel aus ihr, ändert sich nicht.

Ebenfalls ersichtlich ist es, dass q monoton zu Null abnehmend, somit p monoton zum Resultat zunehmend ist.

(Was eigentlich geschieht, ist dies: Der Vektor wird gedreht, bis er mit genügender Genauigkeit horizontal verläuft; dann ist q klein genug, um vernachlässigt zu werden, und p kann als Resultat gelten.)

Daraus folgt die Richtigkeit des Verfahrens.

Weiterhin folgt (aus der Monotonität), dass wenn sowohl die Argumente als auch das Resultat als Zahlengrössen repräsentierbar sind, können die Zwischenresultate nicht überlaufen und nur glimpflich unterlaufen. (Wertet man die Formel $\sqrt{(x^2+y^2)}$ wie geschrieben aus, so geht das nicht durch - für praktisch die Hälfte des Exponentenbereiches gäbe es Über- oder Unterlauf.)

Die relative Abweichung ist

$$(p-\sqrt{(p^2+q^2)})/\sqrt{(p^2+q^2)} = 1/\sqrt{(1+(q/p)^2)}-1.$$

Sie ist im Modul (Absolutwert) durch $(q/p)^2/2$ beschränkt, wie das sich für $q=p$ unmittelbar ergibt und für $q<p$ aus der Binomialreihenentwicklung für den Exponenten $-1/2$ folgt.

Aus den obigen Formeln erhalten wir

$$q^*/p^* = (q/p)^3/(4+3(q/p)^2) \leq (q/p)^3/4,$$

so dass die Konvergenz "überkubisch" ist: die Anzahl der richtig errechneten Stellen wird per Schritt mehr als verdreifacht.

(Nebenbei sieht man, dass $q \leq p$ zwar zur "Überkonvergenz" nötig ist, nicht aber zur Konvergenz schlechthin: Ist $q>p$, so liefert jeder Schritt eine mehr als siebenfache Verkleinerung, bis sich endlich $q\sqrt{p}$ einstellt und die Überkonvergenz Platz greift.)

Der schlimmste relative Fehler ist offensichtlich bei $p=q$. Errechnet man diesen Fall, so stellt es sich heraus, dass nach dem zweiten Iterationsschritt die relative Genauigkeit besser ist als 21 Binärstellen bzw. 6 Dezimalstellen, nach dem dritten besser als 67 Binärstellen bzw. 20 Dezimalstellen, usw.

Das zeigt auch, dass, angesichts der sehr wenigen Iterationen, Fehlerakkumulation kein ernsthaftes Problem ist.

Varianten

Selbstredend braucht man in der Schleife nicht solange verweilen, bis q zu Null unterläuft. Man kann demgegenüber eventuell einen Schritt ganz oder teilweise einsparen. Ist nämlich m der kleinste positive repräsentierbare Modul, so kann man bei $q/p < \sqrt{(2m)}$ getrost abbrechen, da sich das Resultat nicht mehr ändern kann. Moler und Morrison schlagen vor, das als Zwischenresultat sowieso vorhandene $4+r$ als Kriterium wie folgt zu verwenden: Ist $4+r == 4$ gemäss der Rechnerarithmetik, so brechen wir ab. Dies läuft auf dasselbe heraus, ist einfach und kommt ohne Kenntnis von m aus.

Eine Alternative dazu ist, in Anbetracht der kleinen Anzahl nötiger Schritte, die Schleife fixiert-vielmal laufen zu lassen, oder aber sie ganz aufzulösen und ihr Inneres sovielmal als nötig niederzuschreiben. Dadurch verliert man zwar die Möglichkeit, bei q<<p weniger bzw. keine Schritte zu tun. Dieser Verlust ist aber pauschal mehr als aufgewogen durch den Wegfall der organisatorischen Massnahmen. - Man beachte, dass die Iteration im Falle p==0 nicht stattfinden darf.

Für drei Schritte braucht man (höchstens) 29 arithmetische Operationen, davon 12 Multiplikationen und 6 Divisionen, 3 der Multiplikationen können durch einfachere Operationen ersetzt werden. D. h., die Methode lässt sich auch in dieser Hinsicht anschauen.

Als allgemeine Methode ist keine bessere bekannt. Das soll nicht heissen, dass in partikulären Fällen unbedingt diese zur Verwendung kommen muss. Für 16-Bit Fixkomma-Zahlen mit 9-12-Bit Fraktionalteil bevorzugt z. B. Mactor eine Kompromisslösung, die da glücklicher ist. (Vgl. in "Mactor, Matrix/Vector Handling", Budapest 1989, die functions "length" und "lngth3".) Wenn keine grösstmögliche Genauigkeit benötigt ist (und das ist sehr oft der Fall), kann man linear approximieren, das gibt eine Genauigkeit von 4 % mit zwei Multiplikationen und einer Addition; unter Einführung einer vierfachen Intervallteilung kommen zwei bedingte Sprünge dazu und der Höchstfehler sinkt unter 0.5 %. (Vgl. "Vektorhossz, lineárisan", Budapest 1979.)

Für mehr als zwei Dimensionen kann "fleng" selbstverständlich mehrfach angerufen werden, das lässt sich aber auch vermeiden, da das Innere von "fleng" entsprechend geändert werden kann, wobei sich weitere Verbesserungen ergeben.

Modifikationen - Quadratwurzel

Die Methode lässt sich übertragen auf das Berechnen von $\sqrt{(x^2-y^2)}$ (durch Vorzeichenwechsel in der Schleife), von quadratischen Diskriminanten usw. Hier gebe ich nur an, wie sie geändert werden kann, um $\sqrt{x}$ zu errechnen.

Das Programm gestaltet sich jetzt wie folgt:

```
double fsqrt(double x)
{
double p, r, s, t;
  p = 1; r = x-1;
  while (r) { s = r/(r+4); t = 2*s+1; p *= t; s /= t; r*= s*s; }
  return(p);
}
```

x selbst kann als p dienen.

Man kann t aussparen und dabei den Schritt z. B. so gestalten:

```
{ s = r/p; s /= s+4*p; p *= 2*s+1; r *= s*s; }
```

- das ist wohl kürzer, aber nicht schneller.

Was über die Abbruchbedingung oben gesagt worden ist, ist sinngemäss auch hier gültig.

Das Problem ist, wie Moler und Morrison es betonen, dass eine gute Konvergenz nur für x nahe 1 stattfindet. Die Abhilfe ist jedoch einfach, vorausgesetzt, wir haben eine Zahlrepräsentation mit Binärexponenten:

Mit x=0 gibt es keine Schwierigkeit. Sonst ist $x = 2^k*f$ mit f in [1/2, 1). Setzen wir h = [k/2] (Ganzteil von k/2), dann ist $x = 2^{2h}*g$ mit g in [1/2, 2). Die Quadratwurzel ist $2^h*\sqrt{g}$. Die Abtrennung von 2^{2h} und die Einfügung von 2^h sind einfache Exponentmanipulationen; g-1 fällt in [-1/2, 1), wobei die ungünstigsten Werte die Endpunkte sind; rechnet man nach, so hat man auch zahlenmässig dieselbe Konvergenz wie oben. (All das ist freilich unabhängig von der Repräsentation, rein mathematisch definiert; der Binärexponent ist nur erwähnt worden, um die überaus einfache Vollziehbarkeit hervorzuheben.)

Verglichen mit der bekannten Newton-Raphsonschen Iteration $p^* = (p+x/p)/2$, die nur überquadratisch konvergiert, aber weniger Operationen per Schritt gebraucht, ist das Wurzelziehen auf Moler-Morrisonscher Basis im Durchschnitt ungefähr gleich stark; welches Verfahren besser ist, hängt von der Zahlenbreite ab: z. B. für 21 oder 67 Bits ist die neue Methode schneller, im Mittelpunkt dazwischen die alte. (Beachte, dass, um gut zu funktionieren, auch Newton-Raphson einer Initialisation bedarf.) Kommt auch die Programmlänge in Betracht, so gewinnt die alte Methode. Newton-Raphson hat überdies den Vorteil, selbstberichtigend zu sein im Sinne, dass beliebige Fehler im Laufe der weiteren Iterationen wieder verschwinden; Moler-Morrison hat diese Eigenschaft nicht.

Obiges heisst wiederum nicht, dass solche Methoden des Quadratwurzelziehens die schnellsten bekannten Methoden sind. Baut man aus denselben Operationen, woraus auch die Multiplikation aufgebaut werden pflegt, d. h. Ganzzahl-Additionen, Bit-Manipulationen und Verschiebungen, so gibt es einen Algorithmus, der die Quadratwurzel in ungefähr zweieinhalb Multiplikationszeiten ausziehen kann. (Vgl. "Floating-Point Algorithms", third printing, Budapest 1985, S. 17.) Das ist aber anderes Baumaterial. Solche Lösungen eignen sich für Hardware-Bau, bit-slice Prozessoren u. a.; im Software-Bereich scheinen sie nicht konkurrenzfähig zu sein.