Einführung in die Numerik

Peter Bastian

Universität Heidelberg
Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Im Neuenheimer Feld 368, D-69120 Heidelberg
email: peter.bastian@iwr.uni-heidelberg.de

20. Oktober 2016

Zahlen im Computer

Alle Programmiersprachen stellen elementare Datentypen zur Repräsentation von Zahlen zur Verfügung. In C/C++:

`unsigned int, unsigned short, unsigned long`	$\mathbb{N}_0$
`int, short, long`	$\mathbb{Z}$
`float, double`	$\mathbb{R}$
`complex<float>, complex<double>`	$\mathbb{C}$

Diese sind Idealisierungen der Zahlenmengen $\mathbb{N}_0, \mathbb{Z}, \mathbb{R}, \mathbb{C}$.
Bei `unsigned int`, `int` ... besteht die Idealisierung darin, dass es eine größte (bzw. kleinste) darstellbare Zahl gibt. Ansonsten sind die Ergebnisse *exakt*.
Bei `float` und `double` kommt hinzu, dass die meisten innerhalb des erlaubten Bereichs liegenden Zahlen nur *näherungsweise* dargestellt werden können.

Potenzreihe für e^x

- e^x lässt sich mit einer Potenzreihe berechnen:

$$e^x = 1 + \sum_{n=1}^{\infty} \frac{x^n}{n!} = 1 + \sum_{n=1}^{\infty} y_n.$$

- Dies formulieren wir rekursiv:

$$y_1 = x, \qquad S_1 = 1 + y_1, \qquad \text{(Anfangswerte)}.$$

  Rekursion:

$$y_n = \frac{x}{n} y_{n-1}, \qquad S_n = S_{n-1} + y_n.$$

- Probiere verschiedene Genauigkeiten und Werte von x.

Positives Argument

- Für $x = 1$ und `float`-Genauigkeit erhalten wir:

```
1.000000000000000e+00    1   2.000000000000000e+00
5.000000000000000e-01    2   2.500000000000000e+00
1.666666716337204e-01    3   2.666666746139526e+00
4.166666790843010e-02    4   2.708333492279053e+00
8.333333767950535e-03    5   2.716666936874390e+00
1.388888922519982e-03    6   2.718055725097656e+00
1.984127011382952e-04    7   2.718254089355469e+00
2.480158764228690e-05    8   2.718278884887695e+00
2.755731884462875e-06    9   2.718281745910645e+00
2.755731998149713e-07   10   2.718281984329224e+00
0.000000000000000e+00  100   2.718281984329224e+00
                        ex   2.718281828459045e+00
```

  . . . also 7 gültige Ziffern.

- Für $x = 5$ . . .

```
9.331081209830243e-06   21   1.484131774902344e+02
                        ex   1.484131591025766e+02
```

  . . . dito.

Negatives Argument

- Für $x = -1$ und `float`-Genauigkeit erhalten wir:

```
 2.75573198149713e-07    10  3.67879420518875 1e-01
-2.50521097200362 4e-08  11  3.67879390716552 7e-01
 2.08767581000302 0e-09  12  3.67879390716552 7e-01
                         ex  3.67879441171442 3e-01
```

  . . . 6 gültige Ziffern.

- Für $x = -5$

```
-5.00000000000000e+00    1  -4.00000000000000e+00
 1.25000000000000e+01    2   8.50000000000000e+00
-2.08333396911621e+01    3  -1.23333396911621e+01
 2.60416679382324 2e+01   4   1.37083396911621e+01
-2.33372952789068 2e-02  15   1.11889280378818 5e-03
 7.29290489107370 4e-03  16   8.41179769486188 9e-03
 1.22185442319455 7e-10  28   6.73746131360530 9e-03
 0.00000000000000e+00 100   6.73746131360530 9e-03
                         ex   6.73794699908546 7e-03
```

  nur noch 4 gültige Ziffern!

Noch kleineres Argument

- Für $x = -20$ und `float`-Genauigkeit sind ...

```
-2.000000000000000e+01    1  -1.900000000000000e+01
 2.000000000000000e+02    2   1.810000000000000e+02
-1.333333374023438e+03    3  -1.152333374023438e+03
 6.666666992187500e+03    4   5.514333496093750e+03
-2.666666796875000e+04    5  -2.115233398437500e+04
-2.611609750000000e+06   31  -1.011914250000000e+06
 1.632256125000000e+06   32   6.203418750000000e+05
-9.892461250000000e+05   33  -3.689042500000000e+05
 5.819095000000000e+05   34   2.130052500000000e+05
-3.325197187500000e+05   35  -1.195144687500000e+05
 1.847331718750000e+05   36   6.521870312500000e+04
-4.473213550681976e-07   65   7.566840052604675e-01
 1.355519287926654e-07   66   7.566841244697571e-01
-4.046326296247571e-08   67   7.566840648651123e-01
 1.190095932912527e-08   68   7.566840648651123e-01
                         ex   2.061153622438557e-09
```

keine Ziffern mehr gültig. Das Ergebnis ist um 8 Größenordnungen daneben!

Erhöhen der Genauigkeit

- Für $x = -20$ und `double`-Genauigkeit erhält man

```
-1.232613988175268e+07    27  -5.180694836889297e+06
 8.804385629823344e+06    28   3.623690792934047e+06
 1.821561256740375e-24    94   6.147561828914626e-09
-3.834865803663947e-25    95   6.147561828914626e-09
                          ex   2.061153622438557e-09
```

  Immer noch um einen Faktor 3 daneben!
- Erst mit „vierfacher Genauigkeit“ erhält man

```
  118  2.06115362243855833927004587529 47e-09
   ex  2.06115362243855782796594038015 58e-09
```

  15 gültige Ziffern (bei ca 30 Ziffern „Rechengenauigkeit“).

Fragen

- Was bedeutet überhaupt „Rechengenauigkeit“?
- Welche Genauigkeit können wir erwarten?
- Wo kommen diese Fehler her?
- Wie werden solche „Kommazahlen“ dargestellt und verarbeitet?

Obige Berechnungen wurden mit den Paketen `qd` und `arprec` (beide `http://crd.lbl.gov/~dhbailey/mpdist/`) durchgeführt. `qd` erlaubt bis zu vierfache `double` Genauigkeit, `arprec` beliebige Genauigkeit. Die GNU multiprecision library (`http://gmplib.org/`) ist eine Alternative. □