circuits can be given for the computation of the exact scalar product with virtually no unoverlapped computing time needed for the execution of the arithmetic. In a pipeline, the arithmetic can be executed in the time the processor needs to read the data into the arithmetic unit. This means that no other method of computing a scalar product can be faster, in particular not a conventional approximate computation of the scalar product in double or quadruple precision floating-point arithmetic. Fixed-point accumulation of the products is simpler than accumulation in floating-point. Many intermediate steps that are executed in a floating-point accumulation, such as normalization and rounding of the products and the intermediate sum, composition into a floating-point number and decomposition into mantissa and exponent for the next operation, do not occur in the fixed-point accumulation of the exact scalar product. Since the result is always exact, no exception handling is needed.

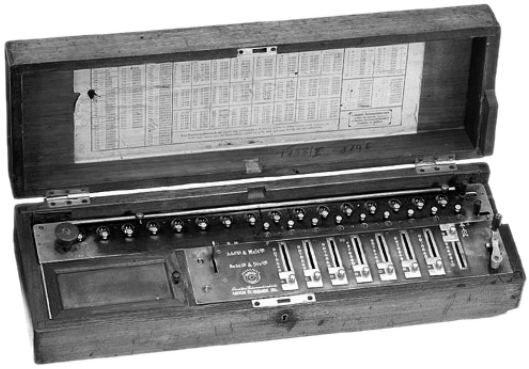
8.2 Historical remarks

Floating-point arithmetic has been used since the early forties and fifties (Zuse Z3, 1941) [63, 554, 555, 588]. Technology in those days was poor (electromechanical relays, electron tubes). It was complex and expensive. The word size of the Z3 was 24 bits. The storage provided 64 words. The four elementary floating-point operations were all that could be provided. For more complicated calculations, error analysis was left to the user.

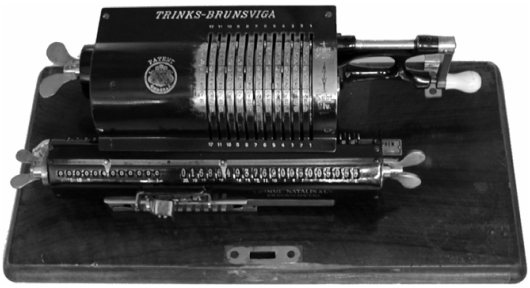
Before that time, highly sophisticated mechanical computing devices were used. Several very interesting techniques provided the four elementary operations addition, subtraction, multiplication and division. Many of these calculators were able to perform an additional *fifth operation* which was called *Auflaufenlassen* or the *running total*. The input register of such a machine had perhaps 10 or 12 decimal digits. The result register was much wider and had perhaps 30 digits. It was a fixed-point register which could be shifted back and forth relative to the input register. This allowed a continuous accumulation of numbers and of products of numbers into different positions of the result register. Fixed-point accumulation was thus error free. See Figures 8.1 (a)–(d).

On all these computers the result register is much wider than the input data register. The two calculators displayed in the Figures 8.1 (c) and (d) show more than one long result register. This allowed the simultaneous accumulation of different long sums.

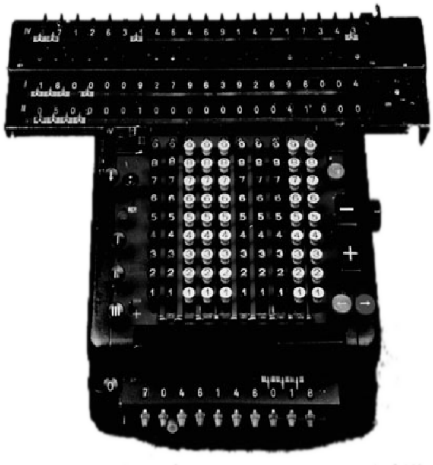
This fifth arithmetic operation was the fastest way to use the computer. It was applied as often as possible. No intermediate results needed to be written down and typed in again for the next operation. No intermediate roundings or normalizations had to be performed. No error analysis was necessary. As long as no underflow or overflow occurred, which would be obvious and visible, the result was always exact. It was independent of the order in which the summands were added. Rounding was only done, if required, at the very end of the accumulation.



(a) **Burkhardt Arithmometer** Step drum calculating machine by Arthur Burkhardt & Cie, Glashütte, Germany, 1878.



(b) **Brunsviga** Pin wheel calculating machine BRUNSVIGA, system Trinks, by Brunsviga Maschinenwerke Grimme Natalis & Co., Braunschweig, Germany, 1917.



(c) **MADAS**, by H.W. Egli, Zürich, Switzerland (1936)
(**M**ultiplication, **A**utomatic **D**ivision, **A**ddition, **S**ubtraction).

(d) **MONROE**, model MONROMATIC ASMD (1956),
by Monroe Calculating Machine Company, Inc., Orange, New Jersey, USA.

Figure 8.1. Some mechanical computing devices developed between 1878 and 1956.

This extremely useful and fast fifth arithmetic operation was not built into the early floating-point computers. It was too expensive for the technologies of those days. Later, its superior properties had been forgotten. Thus floating-point arithmetic is still comparatively incomplete.

After Zuse, the electronic computers of the late forties and early fifties represented their data as fixed-point numbers. Fixed-point arithmetic was used because of its superior properties. Fixed-point addition and subtraction are exact. Fixed-point arithmetic with a rather limited word size, however, imposed a scaling requirement. Problems

had to be preprocessed by the user so that they could be accommodated by this fixed-point number representation. With increasing speed of computers, the problems that could be solved became larger and larger. The necessary preprocessing soon became an enormous burden.

Thus automatic scaling became generally accepted as floating-point arithmetic. It largely eliminated this burden. A scaling factor is appended to each number in floating-point representation. The arithmetic hardware takes care of the scaling. Exponents are added (subtracted) during multiplication (division). It may result in a big change in the value of the exponent. But multiplication and division are relatively stable operations in floating-point arithmetic. Addition and subtraction, on the contrary, are troublesome in floating-point. Early floating point arithmetic was done on early fixed-point machines by programming.

The quality of floating-point arithmetic has been improved over the years. The data format was extended to 64 and even more bits and the IEEE arithmetic standard has finally taken the bugs out of various realizations. Floating-point arithmetic has been used very successfully in the past. Very sophisticated and versatile algorithms and libraries have been developed for particular problems. However, in a general application the result of a floating-point computation is often hard to judge. It can be satisfactory, inaccurate or even completely wrong. Neither the computation itself nor the computed result indicate which one of the three cases has occurred. We illustrate the typical shortcomings by three very simple examples. All data in these examples are IEEE double precision floating-point numbers! For these and other examples see [561]:

Examples. 1. Compute the following, theoretically equivalent expressions:

$$\begin{aligned}
&10^{20} + 17 - 10 + 130 - 10^{20},\\
&10^{20} - 10 + 130 - 10^{20} + 17,\\
&10^{20} + 17 - 10^{20} - 10 + 130,\\
&10^{20} - 10 - 10^{20} + 130 + 17,\\
&10^{20} - 10^{20} + 17 - 10 + 130,\\
&10^{20} + 17 + 130 - 10^{20} - 10.
\end{aligned}$$

A conventional computer using the double-precision data format of the IEEE floating-point arithmetic standard returns the values 0, 17, 120, 147, 137, -10. These errors come about because the floating-point arithmetic is unable to cope with the digit range required with this calculation. Notice that the data cover less than 4% of the digit range of the double precision data format!

2. Compute the solution of a system of two linear equations $Ax = b$, with

$$A = \begin{pmatrix} 64919121 & -159018721 \\ 41869520.5 & -102558961 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The solution can be expressed by the formulas

$$x_1 = \frac{a_{22}}{a_{11}a_{22} - a_{12}a_{21}} \quad \text{and} \quad x_2 = \frac{-a_{21}}{a_{11}a_{22} - a_{12}a_{21}}.$$

A workstation using IEEE double precision floating-point arithmetic returns the *approximate* solution

$$\tilde{x}_1 = 102558961 \quad \text{and} \quad \tilde{x}_2 = 41869520.5,$$

while the correct solution is

$$x_1 = 205117922 \quad \text{and} \quad x_2 = 83739041.$$

After only 4 floating-point operations all digits of the computed solution are wrong. A closer look into the problem reveals that the error happens during the computation of the denominator. This is just the kind of expression that will always be computed exactly by the missing fifth operation.

3. Compute the scalar product of the two vectors a and b with five components each:

$$\begin{aligned}
a_1 &= 2.718281828 \cdot 10^{10}, & b_1 &= 1486.2497 \cdot 10^9,\\
a_2 &= -3.141592654 \cdot 10^{10}, & b_2 &= 878366.9879 \cdot 10^9,\\
a_3 &= 1.414213562 \cdot 10^{10}, & b_3 &= -22.37492 \cdot 10^9,\\
a_4 &= 0.5772156649 \cdot 10^{10}, & b_4 &= 4773714.647 \cdot 10^9,\\
a_5 &= 0.3010299957 \cdot 10^{10}, & b_5 &= 0.000185049 \cdot 10^9.
\end{aligned}$$

The correct value of the scalar product is $-1.00657107 \cdot 10^8$. IEEE-double precision arithmetic delivers $+4.328386285 \cdot 10^9$ so even the sign is incorrect. Note that no vector element has more than 10 decimal digits. IEEE arithmetic computes with a word size that corresponds to about 16 decimal digits. Only 9 floating-point operations are used.

Problems that can be solved by computers become larger and larger. Today fast computers are able to execute several billion floating-point operations every second. This number exceeds the imagination of any user. Traditional error analysis of numerical algorithms is based on estimates of the error of each individual arithmetic operation and on the propagation of these errors through a complicated algorithm. It is simply no longer possible to expect that the error of such computations can be controlled by the user. There remains no alternative to further developing the computer's arithmetic to enable it to control and validate the computational process.

Computer technology is extremely powerful today. It allows solutions which even an experienced computer user may be totally unaware of. Floating-point arithmetic which may fail in simple calculations, as illustrated above, is no longer adequate to be

used exclusively in computers of such gigantic speed for huge problems. The reintroduction into computers of the fifth arithmetic operation, the exact scalar product, is a step which is long overdue. A central and fundamental operation of numerical analysis which can be executed exactly with only modest technical effort should indeed always be executed exactly and no longer only *approximately*. With the exact scalar product all the valuable properties which have been listed in connection with the old mechanical calculators return to the modern digital computer.

> *The exact scalar product is the fastest way to use the computer. It should be applied as often as possible. No intermediate results need to be stored and read in again for the next operation. No intermediate roundings and normalizations have to be performed. No intermediate overflow or underflow can occur. No error analysis is necessary. The result is always exact. It is independent of the order in which the summands are added. Rounding is only done, if required, at the very end of the accumulation.*

This chapter pleads for extending floating-point arithmetic by the exact scalar product as the fifth elementary operation. This combines the advantages of floating-point arithmetic (no scaling requirement) with those of fixed-point arithmetic (fast and exact accumulation of numbers and of single products of numbers even for very long sums). The exact scalar product reintegrates the advantages of fixed-point arithmetic into digital computing and floating-point arithmetic. It is obtained by putting a capability into modern computer hardware which was already available on calculators before the electronic computer came onto the stage.

This chapter claims that, and explains how, scalar products, *whose source data are floating-point numbers*, can always be exactly computed. In the old days of computing (1950–1980) computers often provided sloppy arithmetic in order to be fast. This was "justified" by explaining that the last bit was incorrect in many cases, due to rounding errors. So why should the arithmetic be slowed down or more hardware be invested by computing the best possible answer of the operations under the assumption that the last bit is correct? Today it is often asked: why do we need an exact scalar product? The last bit of the data is often incorrect and a floating-point computation of the scalar product delivers the best possible answer for problems with perturbed data. With the IEEE arithmetic standard this kind of "justification" has been overcome. This allows problems to be handled exactly where the data are exact and the best possible result of an operation is needed. In mathematics it makes a big difference whether a computation is exact for many or most data or for all data! For problems with perturbed (inexact) data, interval arithmetic is the appropriate tool. The exact scalar product extends the accuracy requirements of the IEEE arithmetic standard to all operations in the usual product spaces of computation, to complex numbers, to vectors, matrices, and their interval extensions. If implemented in hardware, it brings an essential speed up of these operations, and it allows an easy and very fast realization of multiple or variable precision arithmetic.