PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 43, NO. 1, PP. 53–64 (1999)

ANALYSIS OF ROUNDOFF NOISE

Rezső DUNAY and István KOLLÁR

Dept. High Voltage Engineering and Equipment
Technical University of Budapest
H–1521 Budapest, Hungary

Received: June 9, 1999

Abstract

All arithmetic operations can be decomposed into an infinitely accurate calculation and a subsequent rounding quantization. The kind of rounding in use determines the properties of the whole arithmetic system. The article discusses tools for the analysis of the effects of rounding errors during arithmetic calculations. The engineering approach is introduced and the applicability of interval arithmetic is also examined for low-precision systems.

Keywords: arithmetic, rounding, quantization, finite bit number, DSP.

1. Introduction

Interval arithmetic is a general tool to evaluate mathematical expressions in order to reach a result with a guaranteed accuracy [6]. This method produces a verified interval, surely containing the exact result. These limits correspond to the 'worst case' estimates in engineering terms.

In cost-effective systems, or systems with low power consumption, usually low bit-number, fixed-point arithmetic is used, even today. The design of software for these systems requires special considerations from the algorithmic point of view (e.g. the optimization of low bit number coefficients), and their validation generally cannot be done with a 100% reliability. In these practical applications the 'worst case' design is simply too expensive (high speed, complexity or cost), in most cases rather statistical methods based on 'confidence intervals' are used. These methods do not guarantee limits to the results, but provide reasonably high-level confidence intervals. In this paper we try to find means for the affordable analysis of such systems.

The paper begins with collecting and comparing some terms, to coin the notation. The next part summarizes some special requirements of practical systems and the main ideas of the engineering approach are described. Examples are given to demonstrate the limitations of the 100% reliability design and also of the engineering approach. The possibility of a common simulation environment for numerical algorithm simulation and analysis is examined in the next two sections, one of them is dedicated to the problems of the MATLAB based implementation [11]. Finally, some conclusions are drawn.

2. Notation

The problem of low bit-number systems stems from the fact that information (i.e. accuracy) is corrupted in every step. This is mainly due to the limited resolution of the destination storage, as in most cases the result has a well-defined and quite low number of bits to represent the resulting value. In engineering practice these quantities, which may have only a finite number of possible values, are called *quantized* quantities. The unit which produces a quantized value from its continuous input is usually referred to as a *quantizer*. Its operation is to round the input to the required accuracy, using the 'rules of rounding' (like number of bits, direction, etc.). Most of today's computers produce the result of all arithmetic operations by rounding the exact result to the destination accuracy, which means that all such arithmetic operations can be decomposed into an infinitely accurate calculation and a subsequent rounding operation. It is straightforward to see that all fixed-point operations can be described by using a simple equidistant quantizer.

There are some new results that analyze floating-point number systems from this aspect [2]. In fact all correctly defined floating-point operations can also be described by a precise operation and a subsequent non-equidistant quantizer (the IEEE Floating-point Standard requires the result of floating point operations to be calculated by rounding the exact result of the operation to the nearest value representable in the accuracy of the actual arithmetic [4]). From the above it can be seen that quantization and finite precision arithmetic lead to the very same problems, and they can be handled in a common framework.

3. Practical Considerations

There are many attributes of engineering work, that originate in limited hardware/software resources or in the nature of the problem to be solved. Two of these aspects will be mentioned here, which – especially together – have a great influence on the arithmetical side of realizations.

- *Feedback* plays a central role in many fields of engineering. It is also an inherent property of all iterative processes: they re-use the result of the previous iteration to calculate the new one. This is one of the most common approaches in engineering science, which can help to speed up systems, to decrease error levels, etc. In practice there are more feedback systems than purely feedforward ones. This is why feedback systems and their analysis are so important for engineers. The simulated examples in the next section all incorporate feedback.
- *Usually very large number of iterations* are necessary. Many digital signal processing systems – like process control units – work 24 hours a day, executing many thousands of steps in each second, processing the results of the

previous steps together with the new incoming data. The systems are usually implemented on digital hardware with arithmetic using no more than 16 bits.

The analysis and validation of the above systems needs special considerations. The examples in the next section demonstrate that at these – otherwise practical – low bit numbers the direct application of interval arithmetic leads to unusable results. To be able to produce useful results the meaning of reliability should be reconsidered.

In practice a reliability of 99% is in most cases satisfactory. System design based on confidence intervals leads to more economical solutions, and there are even cases, when the 100% verified solution is simply not possible.

Confidence calculations are based on modelling the quantization – or roundoff – errors with independent additive noise. The 'amplitude' of this noise is of course connected to the width of the intervals in an interval arithmetic based solution, but the supposed independence and the additive nature of the added noise usually leads to more realistic results. Quantization theory provides analytical ways to handle the effects of these independent noise sources, e.g. the effect of an Analog-Digital converter on the input of a system can be correctly analyzed.

Stable systems have an inherent property which causes injected errors to decay as time, i.e. iterations go on. As quantization errors are modelled to be additive, in case of a stable system their effect should decrease as time passes [10]. This 'forgetting' phenomenon is due to the above special property of all stable systems, and is not taken into account during the direct application of interval arithmetics.

Practical systems are usually too complicated to perform an analytical study, that is why simulation is used most of the time to calculate confidence intervals based on statistical analysis.

Of course the engineering approach has its limitations, too. The main problem is, that the noise model is not always valid and the independence of the separate noise sources cannot always be proven (like in the last example of the next section). There are techniques available (dithering) that aim at assuring the independence of noise components [8], [9].

4. Examples

As a proof to the previous sections, some simulated examples follow. All the examples are realizations of pre-designed Infinite Impulse Response (IIR) digital filters, implemented in a low bit-number arithmetic (12-20 bit). The simulations were carried out using our own quantization functions, interval calculations are also based on correctly implemented quantizers. Some of the demonstrated systems are artificially designed to express a certain symptom, but a digital filter implemented in an IEEE standard is also shown.

To avoid scaling problems, the simulations used floating-point arithmetic, having a 12-20 bit long mantissa, and a 4-bit exponent. The number format used – except for the bit numbers – corresponds to the IEEE standard (hidden leading bit,

sign-magnitude coding, biased exponent). The one using a 12-bit long mantissa occupies 16 bits, so we could also call it a 'half-precision IEEE floating-point number'.

In each example the following simulations were run:

- *IEEE rounding mode:* A simulation was carried out using the above special arithmetic, applying rounding, corresponding to the IEEE standard (round to even) [4]. The input signal and the coefficients were rounded to the lower precision using the IEEE round to even rule. The result is a dotted curve in all figures.
- *Interval arithmetic:* The quantization functions were used to calculate intervals. The input signal and the coefficients – like in the previous case – were rounded to the lower precision using the IEEE round to even rule. Intervals are shown as dashed lines (both the upper and the lower bounds).
- *IEEE double:* As a reference, the systems were also simulated using IEEE double precision arithmetic. The 53-bits of the mantissa in these examples produce an accurate enough result. In this case the input and the coefficients were not rounded. A continuous line is used on the graphs.

In all simulations the state variables of the systems were initialised to zero ($y(k) = x(k) = 0, \text{ if } k < 0$).

Example 1

The first example has a really very simple structure. The 'ideal' system is characterized by the following difference equation:

$$y(k) = -x(k-1) + 0.999y(k-1), \tag{1}$$

where x is the input and y is the output time sequence respectively, and k is the time index.

A slow sine-wave of the form $x(k) = \sin(k/500)$ was used as excitation, and the arithmetic used a 12-bit mantissa. *Fig. 1* shows the simulation results.

The system has a single pole close to the unit circle (+0.999, it is rounded to +0.9990234375 in the 12-bit arithmetic), so from the stability point of view it is close to critical. The intervals plotted with a dashed line show, that the interval gets wide quite fast, as simulation starts, but does not increase without limits. The system in (1) has an absolute feedback greater than 0.999, but definitely smaller than 1, which means that the width of the output intervals may be decreased in every iteration. As the next example will demonstrate this is a very lucky and rare case, because in most systems there are more than one feedback paths.

It is worth to mention here, that the IEEE rounding mode produces quite good results even with a 12-bit mantissa.

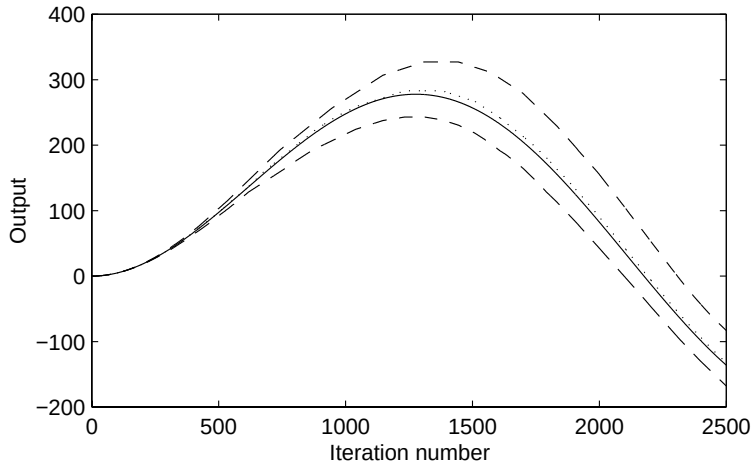

Fig. 1. Simulation results of system (1)

Example 2

The following example is a little bit more complicated than the previous one. Its difference equation is the following:

$$y(k) = -x(k-2) + 1.3y(k-1) - 0.42y(k-2). \tag{2}$$

The notations are the same as in the previous example, the excitation was $x(k) = \sin(k/500)$ also, but the arithmetic used a 20-bit long mantissa. The result can be seen in *Fig. 2*.

This system has two poles (+0.6, +0.7), so from the stability point of view it is quite pleasant. As can be seen in the graph, the width of the interval exponentially grows, and it is already unusable after 40 iteration steps. In the real system the sign of the feedback coefficients is different, which makes the system stable and loosely speaking the output error is decreased in every step (as $1.3 - 0.42 = 0.88 < 1$). In the interval arithmetic realization the output error is represented by the width of the interval. The problem comes from the fact, that in this 'worst case' method the subtract ($-$) operation – similarly to the add operation – adds the width of the intervals of the operands. Loosely speaking the width of the output intervals – the output error – is increased at least by a factor of $1.3+0.42 = 1.72$ in every iteration step. It may be possible that – at least for simple linear systems – a difference equation, similar to the one describing the system can be designed that estimates the width of output intervals. The coefficients of this difference equation depend on the coefficients of the original system and the parameters of the used arithmetics (in case of fixed-point arithmetic it is straightforward, in the case of floating-point arithmetic see [2]) and the stability check of this equation would contain information on the usability of interval arithmetic.

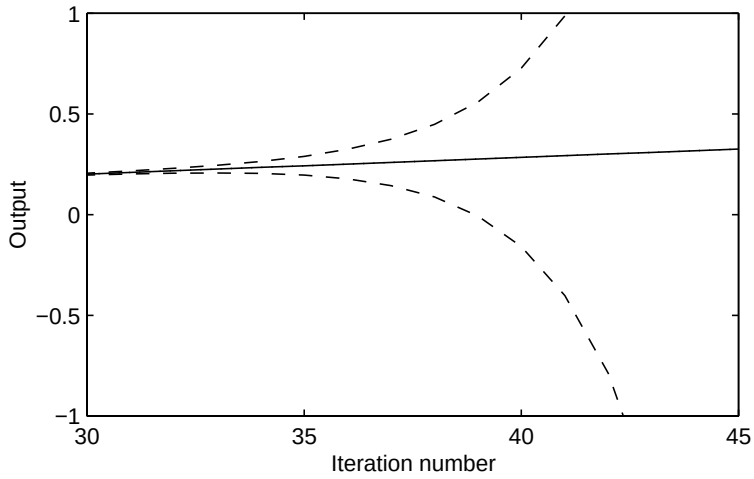

Fig. 2. Simulation results of system (2), with 20-bit long mantissa

Fig. 3 shows the result of the simulation of system (2) on a 12-bit long mantissa arithmetic. As it can be seen, the error of the 12-bit IEEE rounded arithmetic is still not significant after 2500 steps.

Fig. 4 shows the simulation result of system (2) using IEEE double precision interval arithmetic. Though the solid line calculated by the simple IEEE arithmetic was checked to have very small error using analytical methods, the interval calculation leads to exponentially growing intervals (dashed line). This experiment was calculated by the functions defined in [7].

Example 3

Fig. 5 shows a simulation example of an industry standard 1010 Hz Band rejection filter, declared in [1], which has 3 complex pole pairs and 3 complex zero pairs. *Fig. 5* shows the impulse response simulation of this system. Though the simulation used 20 mantissa bits, the intervals get unacceptable within 10 iteration steps.

Example 4

Our last example in *Fig. 6* shows the effect of correlated rounding errors. The figure is actually a magnified part of *Fig. 3* , the dots correspond to the simulation steps of the 12-bit round-to-even arithmetic. The dots situated on horizontal lines mean constant output of the system. In these cases rounding is likely to happen in the same direction, which means that the quantization errors of the subsequent

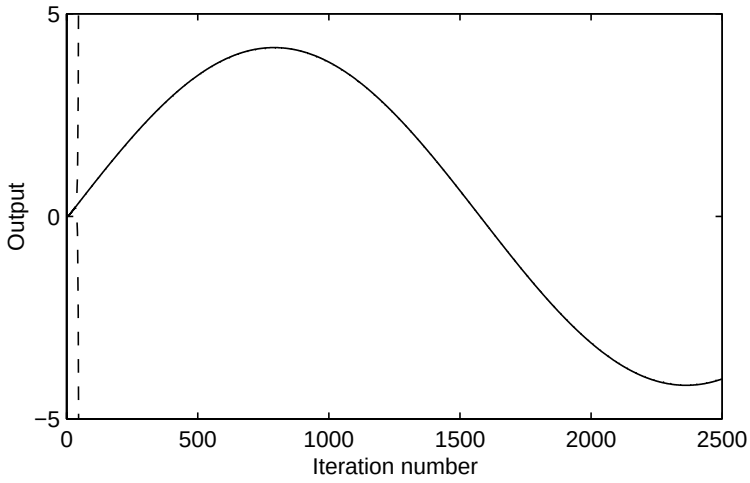

Fig. 3. Simulation results of system (2), with 12-bit mantissa

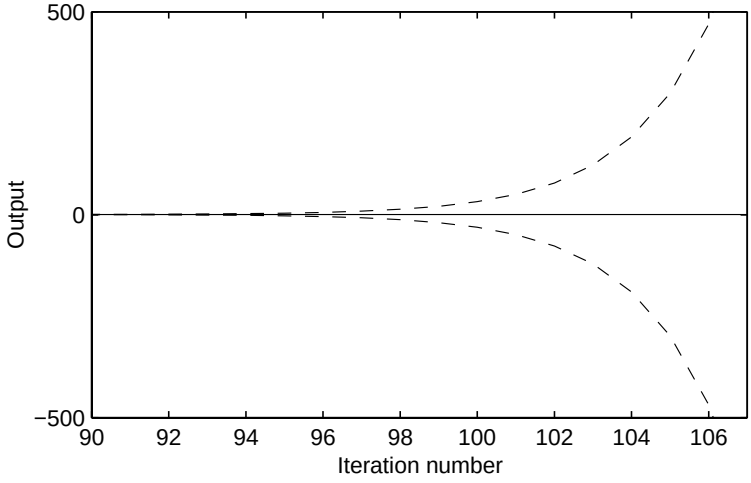

Fig. 4. Simulation results of system (2), with 53-bit mantissa

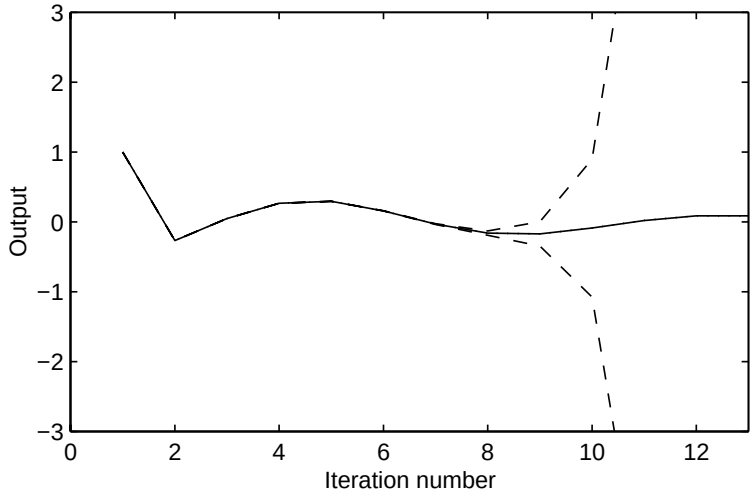

Fig. 5. Simulation of the 1010 Hz Band rejection filter; impulse response

steps are not independent. These cases may belong to fixed-points of the non-linear system, corresponding to the actual excitation.

5. Suggested Simulation Environment

Our goal is to create an environment optimal for testing and analyzing roundoff effects. As explained before, simulation is extremely important in engineering practice. The simulation environment should provide reliable means complying with the following requirements:

- *Reproducibility:* all arithmetic operations should produce a well-defined, reproducible result. It is necessary to be able to certify the experiments [5].
- *Flexibility:* The user should be able to control all aspects of all the operations. The tool should provide functionality to handle all kinds of different systems, including, e.g. the number of bits, rounding modes, etc. The most important quantizer properties implemented in our toolbox are listed in *Table 1*
- *Easy accessibility:* The user should be able to easily access the functionality, express the ideas as close to the mathematical/engineering notation as possible.

The best solution would be to have a single environment capable of simulating the very same program using the native arithmetic of the host computer (this is fast), using interval arithmetic (this provides the verified result) and using a user defined arithmetic (this is needed by the engineers). Running the program with a different precision interval arithmetic is a further option. Fortunately, these options

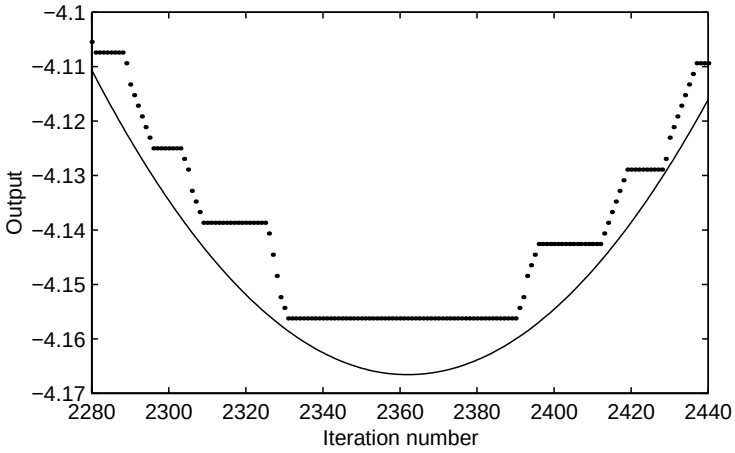

Fig. 6. Correlation of rounding errors

differ only in the underlying arithmetic (type of the data and the operations), but semantically (at the level of the user's program) they can be the same, and can

Table 1. The most important quantizer properties

Property	Description	Values
Operation	Operation mode	quantize, idle, noise model
Type	Type of quantizer	fixed, floating-point
Precision	Number of 'mantissa' bits	1..53
FractBits	Number of fraction bits	1..53
Coding	Coding of the value	one's, two's compl., sign-magn.
RoundDir	Direction of rounding	trunc, round, floor, ceil, toinf
TreatHalf	Handling of .5 LSB values	floor, ceil, fix, toinf, toeven, toodd, rand
Overflow	Type of overflow	clip, modular, triang, inf, none
Emin, Emax	Exponent range	-1023..1022
Underflow	Type of underflow	gradual, flush to zero
ExpCoding	Coding of the exponent	one's, two's compl., sign-magn., biased
ExpBias	Bias of the exponent	
LeadingBit	Leading bit present?	Hidden, present
UfCoding	Coding of underflow	exponent, leading zero
Data	Actual value	
PrecData	Accurate value	

LSB: least significant bit

effectively be implemented in any object oriented programming language. This means that once developed, the same user program could be run in a verified way using high precision interval arithmetics (it can even be higher precision, than the native precision), but a low precision simulation – corresponding to running on a certain Digital Signal Processor – is also possible.

In the following some implementational issues are examined.

There are many software libraries that support interval arithmetic calculations (e.g. [6], [7]). The libraries we know about all support interval arithmetic calculations working only at the native accuracy of the host computer (e.g. IEEE double). Our aim is to analyze lower precision systems, for which we need a tool that is able to execute the calculations working with different precision. It is not difficult to see that lower precision interval arithmetic cannot be properly implemented on top of the available interval arithmetic toolboxes. Complex functions – e.g. SVD – can not be implemented, because lower precision should be used internally by the functions which is usually impossible to achieve. The low level functions, like $+ - */$ cannot be used, because of a phenomenon called re-quantization. The following example is here to demonstrate it. In the example the round-to-even rounding rule is used, *quantization operations* are emphasized.

1.000 1000 1 (9 bits)	*4-bit quantizer*	1.001 (rounded to 4 bits)
8 bit quantizer		
1.000 1000 (rounded to 8 bits)	*4-bit re-quantizer*	1.000 (rounded to 4 bits)

Other direction

1.000 1000 1 (exact result)	*5-bit quantizer*	1.000 1 (rounded to 5 bits)
4 bit quantizer		
1.001 (rounded to 4 bits)	*5-bit re-quantizer*	1.001 0 (extended to 4 bits)

It is clear from the example, that the lower resolution quantization should be executed before the interval arithmetic operation, otherwise the result may be incorrect.

This is why we need a new function library to calculate interval arithmetic working at a lower-than-native accuracy. The above is of course not a fault of the available interval arithmetic toolboxes, but originates in our different goal.

On the other hand, if we had a tool, which is able to simulate arbitrary quantizers, its functionality is enough to calculate interval arithmetic operations, which means that a general quantization toolbox can effectively be used for all the above purposes.

6. Implementation in MATLAB [3]

Engineers use MATLAB very frequently, as its numerical abilities are acceptable, while the description of algorithms is quite easy in its programming language. Its interactive interface makes it very flexible, but requires an interpreted operation. The basic data type in MATLAB is the complex matrix, which helps describing algorithms in a very compact and at the same time easily readable, understandable way. One of MATLAB's other strongholds is the very flexible visualization part. Starting with version 5.0 MATLAB also contains support for Object Oriented Programming, which allows the creation of new data types (class) and redefinition of the underlying operations (operator overloading). From the MATLAB user's point of view it is very useful, because it allows efficient programming by supplying a very algorithm like programming environment (programming time efficiency). The selection of the needed algorithm is not done by ineffective MATLAB program code, but by the effectively programmed MATLAB core, gaining speed (runtime efficiency).

MATLAB seems to be a perfect environment for our purpose. It is possible to define a new data type for quantized data (quantized data class), instances of this class can store quantized elements, together with the parameters of the used quantizer, and the functionality belonging to our new data type can also be defined.

MATLAB itself is a homogeneous system in the sense that only a single floating-point data type (usually the IEEE double precision) is applied. An interval arithmetic toolbox in MATLAB naturally uses this built-in data type. Practical systems, however, frequently use mixed accuracy data (e.g. data types in programming languages, higher precision accumulator in DSP and common microprocessors), so our simulation environment should be able to handle different accuracy data types also. Accuracy of the data itself is a property contained in the quantized variable itself. As MATLAB is an interpreted language (and we use an object oriented approach) the type of the result of a mixed precision operation is not defined by the operation, but by the accuracy of the operands. In our system the accuracy of the result is determined by the first quantized type input parameter from left to right.

Another important question concerns data conversion. The problem is that re-quantization should be avoided during data accuracy conversions. In case of conversion to a higher precision there are two possible approaches:

1. The value should not change.
2. The value should be the exact result of the previous operation, quantized only by the new quantizer.

In case of conversion to a lower precision there are also two possibilities:

1. Simply quantize the value by the new quantizer (re-quantization).
2. The value should be the exact result of the previous operation, quantized only by the new quantizer (avoid re-quantization).

The 2nd variant of conversions in both directions needs storing of the exact value of the previous operation. In general, storing the exact value is not possible due

to limited resources, but for an exact re-quantization the exact value is fortunately not really needed. Only the necessary number of bits and some extra information about the remainder is necessary (direction of the roundoff error and its value compared to .5 ULP).

7. Conclusions

If we want to use the very same environment for the simulation of systems using interval arithmetic, user defined number formats and the combination of the previous two, then a new software environment is needed. MATLAB is a very promising candidate for this purpose.

Simulation results generated using our quantization toolbox show, that direct application of low precision interval arithmetic leads to unusable results in most cases. This is due to the special architecture of many practical engineering systems. A 100% reliability interval arithmetic based design would be interesting for engineers, but serious refinements are needed.

References

[1] IEEE Standard Equipment Requirements and Measurement Techniques. ANSI/IEEE Standard 734-1985, New York, NJ, Apr. 1985.

[2] KOLLÁR, I.:'Statistical Theory of Quantization,' Doctoral Thesis, Hungarian Academy of Sciences, Budapest, 1996, p. 416.

[3] MATLAB User's Manual, Version 5.2, The MathWorks, Inc., Natick, MA, 1998.

[4] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Standard 754-1985, New York, NJ, Aug. 1985.

[5] KAHAN, W.: 'The Baleful Effect of Computer Benchmarks upon Applied Mathematics, Physics and Chemistry", 1995.
http://http.cs.berkeley.edu/~wkahan/ieee754status/baleful.ps

[6] HAMMER, I. – HOCKS, M. – KULISCH, U. – RATZ, D. "C++ Toolbox for Verified Computing", Springer-Verlag, 1995.

[7] RUMP, S. M.: "From INLAB to MATLAB", *SCAN-98 IMACS/GAMM International Symposium on Scientific Computing*, Computer Arithmetic and Validated Numerics, Budapest, Hungary, 22-25. Sep., 1998, Text submitted to the special issue of Reliable Computing.

[8] DUNAY, R. – KOLLÁR, I. – WIDROW, B.: 'Dithering for Floating Point Number Representation', *Dithering in Measurement: Theory and Applications, 1st International On-Line Workshop*, Prague, Czech Republic, Feb-Mar, 1998.

[9] *Dithering in Measurement: Theory and Applications, 1st International On-Line Workshop*, Edited by Holub, J., Smid, R., Prague, Czech Republic, Mar. 1998, http://measure.feld.cvut.cz/dithering98/

[10] BERTRAM, J. E.: "The Effect of Quantization in Sampled-Feedback Systems", *Trans. AIEE*, **77**, pt. 2, pp. 177–82, 1958.

[11] DUNAY, R. – KOLLÁR, I.: "MATLAB-Based Analysis of Roundoff Noise" in Csendes, T. (ed.): Developments in Reliable Computing, Kluwer, Dordrecht, pp. 373-382, 1999.