Using the Variable-Length Arithmetic for an Accurate Poles-Zeros Analysis

Josef DOBEŠ, Jan MÍCHAL

Dept. of Radio Electronics, Czech Technical University, Technická 2, 166 27 Praha, Czech Republic

dobes@fel.cvut.cz, michal@fel.cvut.cz

Abstract. *In the paper, a reduction algorithm for transforming the general eigenvalue problem to the standard one is presented for both classical full-matrix methods and a sparse-matrix technique appropriate for large-scale circuits. An optimal pivoting strategy for the two methods is proposed to increase the precision of the computations.*

The accuracy of the algorithms is furthermore increased using longer numerical data. First, a long double precision sparse algorithm is compared with the double precision sparse and full-matrix ones. Finally, the application of a suitable multiple-precision arithmetic library is evaluated.

Keywords

Poles-zeros analysis, sparse-matrix reduction, general eigenvalue problem, multiple-precision arithmetic.

1. Introduction

The poles-zeros analysis is indisputably among the most important parts of design of electronic circuits. However, the programs of the PSpice family do not have such algorithms implemented (the Spice3 and HSpice programs only have this capability). An original software tool called CIA (Circuit Interactive Analyzer) is frequently used for the poles-zeros analysis for this reason.

The poles-zeros analysis is known to be very sensitive to the numerical precision of algebraic operations during the process. Consequently, many of the theoretically exact methods fail, especially for large-scale circuits.

Two major types of improvements to these methods are proposed here:

- one consists in a meticulous algorithm design regarding the choice of pivots,
- and the other is based on using longer numerical data types together with more precise arithmetic – either as just fully utilizing the given hardware capabilities or by applying a suitable multiple-precision arithmetic library.

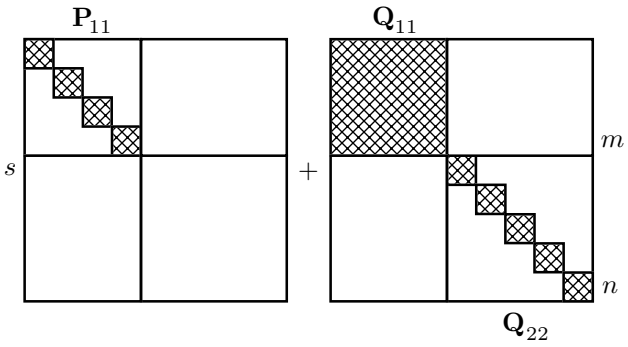

Fig. 1. Final shapes of the matrices after the reduction.

2. Algorithms

2.1 Definition of Reduction Algorithms

A system of linear equations (or equations linearized at an operating point) of a circuit can be written by means of Laplace transformation

$$(s\mathbf{P} + \mathbf{Q})\mathbf{X} = \mathbf{Y}, \tag{1}$$

where s denotes the Laplace operator, $\mathbf{P}/\mathbf{Q}$ are the matrices associated with the dynamic/static parts of the model derivatives, respectively, $\mathbf{X}$ is the vector of Laplace transforms of circuit variables, and $\mathbf{Y}$ is the source vector.

The poles of all the transfer functions and the zeros of a transfer function can be computed solving the equations

$$\begin{aligned} \det(s\mathbf{P} + \mathbf{Q}) &= 0 \ \text{ for poles,} \\ \det\left(s\mathbf{P}^{(0j)} + \mathbf{Q}^{(ij)}\right) &= 0 \ \text{ for zeros,} \end{aligned} \tag{2}$$

where matrices $\mathbf{P}^{(0j)}$ and $\mathbf{Q}^{(ij)}$ arise from the original ones – the first by clearing j^{th} column and the second by replacing j^{th} column by $\mathbf{Y}$ with all its elements cleared with the exception of the element corresponding to the i^{th} source.

Solving the general eigenvalue problem defined by (2) is more difficult than solving the standard one. Therefore, a systematic reduction is applied during transforming (2) to the standard form, which is shown in Fig. 1 (it is a variation of the method in [1]). After the transformation, the determinant can be computed in the classical way

$$\det(s\mathbf{P}+\mathbf{Q}) =$$
$$(-1)^{n_{\text{exch}}} \prod_{i=m+1}^{n} Q_{22_{ii}} \det(\mathbf{P}_{11}\mathbf{P}_{11}^{-1}(s\mathbf{P}_{11}+\mathbf{Q}_{11})) = \quad (3)$$
$$(-1)^{n_{\text{exch}}} \prod_{i=1}^{m} P_{11_{ii}} \prod_{i=m+1}^{n} Q_{22_{ii}} \det(s\,\mathbf{1}+\mathbf{P}_{11}^{-1}\mathbf{Q}_{11}),$$

where n_{exch} is the total number of row and column exchanges while reducing, and $\mathbf{1}$ is the unity matrix. The operations that transform the matrix $s\mathbf{P}+\mathbf{Q}$ to the form drawn above are a certain modification of Gauss elimination method. The only exception occurs when the matrix $\mathbf{P}_{11}$ contains a nondiagonal element that is not reducible by the diagonal elements of this matrix. In such cases, it is necessary to multiply some row from the lower part of the matrix by the s operator. It is equivalent to moving a row of the $\mathbf{Q}_{22}$ matrix left. The nondiagonal element of the $\mathbf{P}_{11}$ matrix can readily be reduced by means of the transferred row. Note that the two products in the equation (3) may be enormous for large-scale circuits and therefore only their signs and logarithms may be stored.

2.2 Implementation of Reduction Algorithms

The reduction process is very difficult from the point of view of numerical precision in the case of large-scale systems. The matrices often contain elements of various magnitudes. Therefore, a *full pivoting* is to be used for the choice of the k^{th} key element

$$P_{kk} \leftarrow \max_{i=k,j=k}^{n,n} |P_{ij}|,\ k \in \{1,\ldots,n\}, \quad (4)$$

however, the key element determined in the above way is regarded to be zero if it is too small in comparison with the greatest element of the k^{th} column

$$\text{if } |P_{kk}| \le \varepsilon_{\text{eigen}} \max_{i=1}^{k-1} |P_{ik}|, \text{ then } P_{kk} := 0. \quad (5)$$

$\varepsilon_{\text{eigen}}$ is an important parameter of the reduction algorithm. Inappropriately large value of this parameter causes ignoring some (real, in fact) poles or zeros, inappropriately small value of the parameter causes computing superfluous (spurious, in fact) poles or zeros (about 10^{-15} is recommended).

The key elements determined in the upper and lower parts of the matrices are used for the reduction of remaining elements of the matrices if they are not too small

$$\text{if } |P_{i'j'}| \le \varepsilon_{\text{round}} \max_{i=k,j=k}^{n,n} |P_{ij}|, \text{ then } P_{i'j'} := 0, \quad k = 1,\ldots,n-1,\ i' \ge k \wedge j' \ge k, \quad (6)$$

$$\text{if } |Q_{i'j'}| \le \varepsilon_{\text{round}} \max_{i=m+1,j=m+1}^{k,k} |Q_{ij}|, \text{ then } Q_{i'j'} := 0, \quad k = n,\ldots,m+1,\ m < i' \le k \wedge m < j' \le k. \quad (7)$$

$\varepsilon_{\text{round}}$ is a second parameter of the algorithm. It prevents the reduction of small elements that can arise because of rounding errors. When such elements are reduced, the number of nonzero elements of the matrices increases and the total error increases too (about 10^{-20} is recommended).

The reduction algorithm is to be implemented using the sparsity of the matrices $\mathbf{P}$ and $\mathbf{Q}$. These matrices are sparse enough already for not very complicated tasks. However, application of the full pivoting is then very difficult from the programming point of view. Therefore, only *partial pivoting* must be used here – the k^{th} key element is chosen from the rest of the k^{th} column of a reduced matrix

$$P_{kk} \leftarrow \max_{i=k}^{n} |P_{ik}|,\ k \in \{1,\ldots,n\}, \quad (8)$$

however, the key element determined in the above way is regarded to be zero if it is too small in comparison with the greatest element of the k^{th} row

$$\text{if } |P_{kk}| \le \varepsilon \max_{j=k+1}^{n} |P_{kj}|, \text{ then } P_{kk} := 0. \quad (9)$$

A final step for determining the poles and zeros of transfer function is naturally computing the eigenvalues of the matrix (i.e., solving the standard eigenvalue problem)

$$\mathbf{Q}' = -\mathbf{P}_{11}^{-1}\mathbf{Q}_{11} \quad (10)$$

by the double-step QR algorithm with automatic shift of origin [2, 5].

2.3 Design of Variable-Length Arithmetic

The arbitrary precision (variable-length) arithmetic routines have been implemented in the Pascal language. Since the design was made with portability in mind, the ISO 7185 standard was strictly obeyed. In addition, only a subset of the language common with Borland Pascal/ Delphi was used. Simplicity and clarity of the design encouraged by the chosen programming language are considered to be major virtues. The variable-length natural numbers representing the mantissa parts are implemented by means of dynamically allocated linked lists rather than arrays. This eliminates the danger of undesirable heap fragmentation, which could otherwise cause allocation failure before all available memory has been used. Only the classical general algorithms described in [3] have been employed for the four basic arithmetic operations. Since the results of floating point operations are by principle approximate (no matter how long the mantissa has been chosen), an optional mechanism has been added maintaining upper estimates of the cumulated roundoff errors for each variable. This can be useful whenever the information about the *guaranteed* accuracy of obtained results is needed. No special optimization to enhance the execution speed has yet been performed. This is supposed to be part of the prospective next stage of development, finally resulting in

partial or complete conversion of the routines into the assembly language, utilizing all technical capabilities of the given hardware.

The implementation of the variable-length floating point (VLFloatingPoint) arithmetic is part of a more extensive library of routines, covering a hierarchy of other variable-length numerical types: natural (or nonnegative integer) VLNatural, integer VLInteger, rational VLRational and complex VLComplex. All these types adopt the same philosophy of use: every variable of a particular type X (standing for any of N, I, R, or F) has first to be allocated by the VLXNew procedure. If it is to be used as an input to an arithmetic operation (VLXAdd, VLXSub, VLXMul, or VLXDiv), it needs to be initialized by VLXInit. By the end of a computation, all used variables should be deallocated by VLXDispose. Values can be converted from/into the standard real type by VLXFromReal and VLXToReal, and read from / written into text files by VLXRead and VLXWrite, respectively (except the VLFRead routine, which has yet to be implemented). For the programmer's convenience, all arithmetic procedures are designed to allow for variables to be given simultaneously as input as well as output parameters.

Calling Pascal procedures from inside C code of the CIA algorithms has been made possible using the GNU family of compilers (specifically, its DJGPP port for MS-DOS), for GNU Pascal has types and calling conventions compatible with those of GNU C.

3. Results

3.1 Improvement of Sparse Mode With "long double" Arithmetic

Let us discuss an AB-class power operational amplifier in Fig. 3 for which reduced [4, 6] (cuts "similar" poles/zeros) poles-zeros configuration is presented in Fig. 2. It is a very suitable example for checking the precision of the sparse mode of reduction.

In general, the full-matrix version of the reduction algorithm seems to work sufficiently. However, its numerical precision is still inadequate for a certain class of tasks solved by the sparse-matrix one. For that reason, a long double ($\geq$10 bytes) version of the sparse-matrix reduction has been developed.

In Tab. 1, a comparison between sparse-matrix double and sparse-matrix long double reductions is performed. Considering the results with 1024bit-mantissa variable-length arithmetic (see the following subsection) to be correct, the incorrect digits are marked by "strikethrough". As observed, the poles have been computed quite precisely. However, the accuracy of the computation of zeros is considerably worse – it is caused by the difference between the smallest and largest pole/zero magnitudes, respectively.

For the multiplying constant of transfer function, the inaccuracy is similar. All the results indicate that the long double sparse-matrix reduction introduces a welcome improvement. Still, the results are only correct to three decimal places. The results with the full-matrix reduction are more precise with the exception of zero at the origin (the full-matrix algorithms do not implement the special structure of matrices used for a "smart" elimination of zeros at the origin by the sparse algorithm). However, the full-matrix reduction cannot be used for large-scale circuits.

Therefore, the application of unlimited- or variable-length arithmetic is expected to perform robust and accurate analyses.

3.2 Improvement of Sparse Mode With Variable-Length Arithmetic

Results obtained with the sparse-matrix reduction algorithm rewritten to call the variable-length arithmetic routines are presented in Tab. 2.

The parameter ε was chosen to be 10^{-23}, i.e., the same as in the long double computation, to allow to compare the corresponding results.

With mantissa length limited to 64 bits, which is the same length as in the long double type (Extended Precision of IEEE 754), the achieved precision of results is basically the same as with long double. A slight tendency towards the correct values is already visible in the variable-length case due to more correct rounding strategy used.

All the poles and zeros obtained with 128bit mantissa are already correct to 6 decimal places. The results file differs from the ones for longer mantissas (256, 512 and 1024 bits were tried) only in the ordering of zeros done by the procedure solving the standard eigenvalue problem. All the poles and zeros for 256 bits and more came out the same

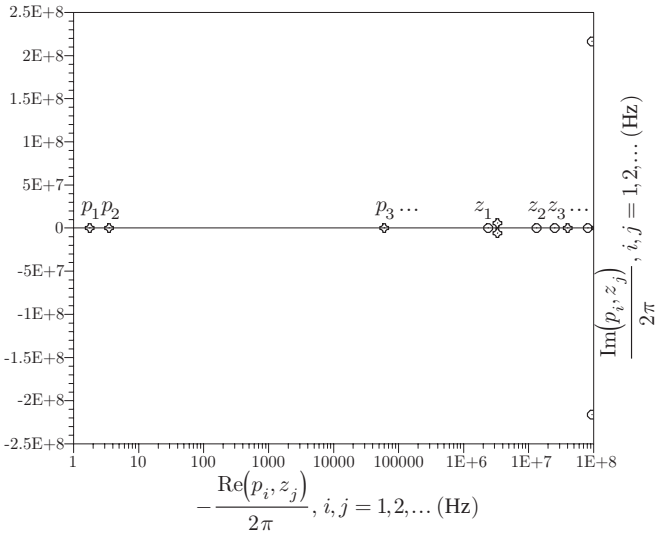

Fig. 2. Reduced poles-zeros (all divided by 2π) diagram of the linearized AB-class power operational amplifier.

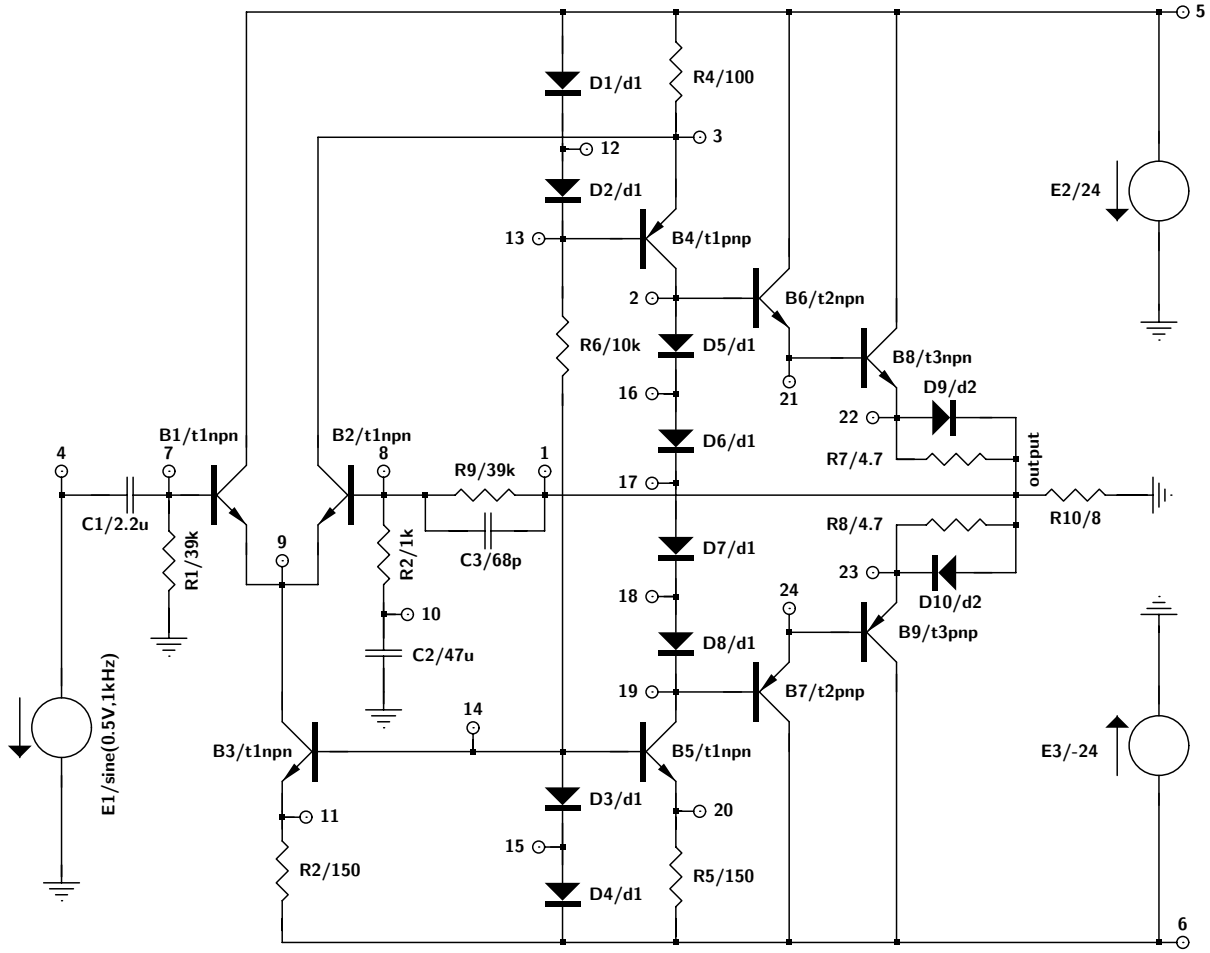

Fig. 3. AB-class power operational amplifier with wide spread poles and zeros on which all the reduction techniques have been compared.

Crucial elements of transfer function regarding exactness	Full-matrix algorithm with $\varepsilon_{eigen}=10^{-15}$, $\varepsilon_{round}=10^{-20}$	Sparse-matrix *double* algorithm with $\varepsilon=10^{-19}$	Sparse-matrix *long double* algorithm with $\varepsilon=10^{-23}$
Zero which should be 0 Hz	-0.986035×10^{-6} Hz	0 Hz	0 Hz
Smallest pole by magnitude	−1.76518 Hz	−1.76518 Hz	−1.76518 Hz
Biggest pole by magnitude	-7.31331×10^{10} Hz	-7.31330×10^{10} Hz	-7.31331×10^{10} Hz
`Smallest´ zero by magnitude	−0.0847816 Hz	−0.0847825 Hz	−0.0847825 Hz
Biggest zero by magnitude	-6.99856×10^{10} Hz	-7.04891×10^{10} Hz	-6.99916×10^{10} Hz
Const. of transfer function	0.988898	1.04496	0.988968

Tab. 1. Comparison of the crucial elements of the amplifier transfer function obtained using *fixed-length* compiler's arithmetic procedures. The control parameters have been optimized so that the results be closest to those obtained with 1024bit-mantissa variable-length arithmetic.

The same monitored elements	64bit mantissa, $\varepsilon=10^{-23}$	128bit mantissa, $\varepsilon=10^{-23}$	256bit mantissa, $\varepsilon=10^{-23}$
Zero which should be 0 Hz	0 Hz	0 Hz	0 Hz
Smallest pole by magnitude	−1.76518 Hz	−1.76518 Hz	−1.76518 Hz
Biggest pole by magnitude	-7.31331×10^{10} Hz	-7.31331×10^{10} Hz	-7.31331×10^{10} Hz
`Smallest´ zero by magnitude	−0.0847825 Hz	−0.0847825 Hz	−0.0847825 Hz
Biggest zero by magnitude	-6.99775×10^{10} Hz	-6.99856×10^{10} Hz	-6.99856×10^{10} Hz
Const. of transfer function	0.988965	0.988898	0.988898

Tab. 2. Comparison of the same transfer function elements obtained using suggested *variable-length* arithmetic procedures. The results for the 128bit and 256bit mantissas are equal regarding the first six valid digits (the poles and zeros are only ordered in slightly different ways).

to 6 decimal places and in the same order of listing in the results file. The duration of the computations did not exceed approximately ten minutes even for the 1024-bit mantissa length.

Changing the ε value was also tried for mantissas of 256 and 1024 bits. For 256 bits, it turned out to be possible to use its value as tiny as 10^{-150} without any change in the highest six digits and order in the results file. With 1024 bits, this limit even drops below 10^{-320}.

Experiments with the rational "unlimited-precision" arithmetic were also carried out; the computing complexity, however, turned out to be too high for the present example. The poles computation was interrupted after about three days, when only about a half of the approximate total of 200,000 arithmetic operations had been finished. And since the duration of multiplications and divisions increases with about the square of operand length, it is virtually impossible to estimate the time needed for the second hundred thousand arithmetic operations.

The extent of numerical misconditioning of this task can well be illustrated by the fact that during the carried-out part of the "unlimited-precision" computations, subtractions of different values equal to even more than 30 decimal digits were performed.

4. Conclusion

As expected, using the long double version of the sparse-matrix reduction provides an unquestionable improvement of the poles-zeros precision. However, there exist some examples for which even the long double precision is insufficient. In those cases, using the variable-length arithmetic is necessary, which can thus be considered to be the ultimate solution of the poles-zeros precision problem.

Acknowledgments

This paper has been supported by the Grant Agency of the Czech Republic, grant No 102/01/0432 (the algorithmic part), and by the Research Program No J04/98:212300014 of the Czech Technical University in Prague (the circuit modeling part).

References

[1] RÜBNER-PETERSEN, T. On sparse matrix reduction for computing the poles and zeros of linear systems, In *Proceedings of the 4th International Symposium on Network Theory*. Ljubljana (Slovenia), 1979.

[2] RALSTON A. *A first course in numerical analysis*. New York: McGraw-Hill Inc., 1965.

[3] KNUTH, D. *Seminumerical algorithms*. 3rd ed., vol. 2 of The Art of Computer Programming. Reading (MA): Addison-Wesley, 1997, sections 4.2.1 and 4.3.1.

[4] BIOLEK, D. Secondary root polishing techniques in computer circuit analysis. In *Proceedings of the Radioelektronika'99.* Brno (Czech Republic), 1999, p. 42 - 45.

[5] DOBEŠ, J. An accurate poles-zeros analysis for large-scale analog and digital circuits. In *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems.* St.Julians (Malta), 2001, p. 1027 - 1030.

[6] BIOLEK, D., BIOLKOVÁ, V., DOBEŠ, J. (Semi)symbolic modeling of large linear systems: pending issues. In *Proceedings of the ISSSE'01 URSI International Symposium on Signals, Systems, and Electronics.* Tokyo (Japan), 2001, p. 397 - 399.

About Authors...

Josef DOBEŠ received the Ph.D. degree in microelectronics from the Czech Technical University, Prague in 1986. From 1986 to 1992, he was a researcher of the TESLA Research Institute, where he performed analyses on algorithms for CMOS Technology Simulators. Currently, he is with the Department of Radio Electronics of the Czech Technical University. His research interests include the physical modeling of radio electronic circuit elements, especially RF and microwave transistors and transmission lines, creating or improving special algorithms for the circuit analysis and optimization, such as time- and frequency-domain sensitivity, poles-zeros or steady-state analyses, creating a comprehensive CAD tool for the analysis and optimization of RF and microwave circuits.

Jan MÍCHAL was born in 1969. He received his M.Sc. degree in electrical engineering from the Czech Technical University in Prague in 1993. Currently he works as a teaching assistant at the Department of Radio Electronics, Faculty of Electrical Engineering, CTU, where he also studies in the Ph.D. program. His research interests mainly consist in the numerical methods for RF circuit analysis and synthesis, especially optimization methods.